PhD-FSTC-2011-03
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on 23/02/2011 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Ivica NIKOLIC
Born on 6 November 1981 in Kumanovo (Macedonia)

CRYPTANALYSIS AND DESIGN OF SYMMETRIC PRIMITIVES

Dissertation defense committee

Dr Alex Biryukov, dissertation supervisor
Professor, Université du Luxembourg

Dr Bart Preneel
Professor, Katholieke Universiteit Leuven, Belgium

Dr Jean-Sébastien Coron, Chairman
Professor, Université du Luxembourg

Dr Joan Daemen
STMicrolectronics, Zaventem, Belgium

Dr Volker Müller, Vice Chairman
Professor, Université du Luxembourg

To my mother,
for her love, wisdom and support

Acknowledgements

First, I would like to thank my supervisor, Prof. Alex Biryukov who has taught me the fine art of cryptographic thinking – from the very small details and nuances to the very large general picture. Alex has done so with a great amount of guidance, encouragement, patience and support. He has provided me a plenty of research ideas and given me a freedom to explorer on my own, while helping with crucial remarks on the way to obtaining the final results.

I thank Prof. Jean-Sébastien Coron, Dr Joan Daemen, Prof. Volker Müller and Prof. Bart Preneel, for reading this manuscript and serving as jury members on my PhD thesis committee.

My gratitude goes to the fellow cryptographers and my co-authors: Alex Biryukov, Dmitry Khovratovich, Krystian Matusiewicz, María Naya-Plasencia, Josef Pieprzyk, Christian Rechberger, Arnab Roy, Yu Sasaki, Martin Schläffer, Ron Steinfeld, Przemysław Sokołowski and Ralf-Philipp Weinmann.

It was a privilege to work with some extraordinary people at LACS. I would like to thank Dmitry Khovratovich for the hours and hours of mental exercise and out-of-the-box thinking, Ilya Kizhvatov for the huge amount of helpful advices and Ralf-Philipp Weinmann for sharing his incredible diversity of knowledge. For many scientific and not so scientific joyful discussions, ping-pong, basketball and pool sessions, fun and unforgettable moments, group trips and parties, I thank my friends and colleges at Uni.Lu: Aivar, Ania, Anna, Arnab, Avradip, Bin, David, Dima, Deike, Edik, Eli, Gaëtan, Ilya, Jean-François, Johann, Kuba, Ma, Milan, Ola, Olya, Pepo, Ralf, Raya, Serzh and Vanya. Special thanks goes to our secretaries Fabienne, Ragga, Mireille and Isabelle for making all the administrative work almost invisible.

I have spent three wonderful months on an internship at the Macquarie University in Sydney. I thank Josef Pieprzyk for hosting me, as well as Ron Steinfeld and Przemysław Sokołowski for making the stay pleasant.

Through the whole PhD program I have been provided a great financial support by Fonds National de la Recherche (FNR) Luxembourg. Their assistance is greatly acknowledged.

Finally, I would like to thank the people who mean world to me: my brothers Aleksandar and Antonio, and my mother Fidanka. It is easy to give your best when you have such a wind at your back.

Ivica Nikolić, February 2011

Abstract

Cryptographic primitives are the basic building blocks of various cryptographic systems and protocols. Their application is based on their well established properties. The security of a crypto system is proven under the assumption that the underlying cryptographic primitives provide some specific security levels. Therefore it is critical to use primitives that can meet these requirements. However, there is no general approach of constructing fast and provably secure primitives. Rather, the primitives undergo years of thorough cryptanalysis and only after no attacks have been found, they can be considered for real world applications.

This PhD thesis deals with the two main cryptographic primitives: block ciphers and cryptographic hash functions. The main contribution lies in presenting attacks on these algorithms. The analysis ranges from finding ad-hoc differential trails that are used for collision search and distinguishers on specific hash functions to automatic search tools that give the optimal differential trails for block ciphers. Weaknesses are shown for a number of SHA-3 candidates in the framework of rotational distinguishers and meet-in-the-middle based preimage attacks.

Contents

List of Figures

List of Tables

Part I

Introduction

The information era has started in the second half of the twentieth century and has stimulated the birth of many scientific disciplines. Cryptology was created to satisfy one of the basic demands of the information expansion. It is the science of securely sharing information and data. Some of the concepts of cryptology predate the information era. The ciphers, probably the most popular abstractions of cryptology, were first used to securely communicate data long before any information science was known. It is a well documented fact that Caesar used ciphers to send and receive messages with his generals.

Our modern technological society heavily relies on information interchange. Whenever the data has to be communicated securely, cryptology is instantiated. To understand the main contribution of cryptology, the following model is introduced. Two parties, sending and receiving[1], want to communicate, i.e. want to exchange messages and data. However, the communication channel that they use is insecure, i.e. there is a malicious party, further called *an adversary* or *an attacker*[2], who can possibly eavesdrop or even alter all the communicated data. The main task of cryptology is to provide algorithms for the sending and receiving parties, such that the adversary cannot obtain any information about the communicated data, nor can meaningfully influence or alter this communication other than erasing the data.

The main applications of cryptology include the leading force of the information exchange – the internet, as well as various communication systems such as phones, ATM machines and others. However, it is substantial to understand that cryptology is not only about enciphering data – the area of application of cryptology is much wider. Usually, its objectives are divided into four main fields:

- *Confidentiality* - keeping the data secret to all parties except for the intended receivers. This goal is achieved by enciphering the data using a secret key which is shared only among the two parties. The enciphered data is unintelligible to anyone who does not have the secret key, hence the information is confidential. For example, confidentiality plays a crucial role in exchanging secret messages in the military – a weakness in the enciphering algorithm might lead to a serious military loss[3].

- *Data integrity* - maintaining the data unaltered. This mechanism provides the ability to detect manipulation of the original data in a form of insertion, deletion and substitution of some data characters. This is very useful in some particular applications where it is much more important to have the data unaltered than confidential. For example, it is crucial for a bank transaction statement to maintain its integrity (though it does not have to be secret). Otherwise, an adversary would be able to change the transaction amount.

- *Authentication* - identification of the parties that communicate and of the communicated data. Each of the parties wants to be sure that the other party is authentic (it is the one it claim it is), and that all the received data is from

[1] Often in cryptology these parties are called Alice and Bob.

[2] In cryptology this party is called Eve.

[3] It has been speculated that one of the greatest British victories in WWII was actually the break of the German cipher machine Enigma.

the authenticated party. Otherwise, an adversary can pretend to be a trusted party and obtain some type of benefit from the other unsuspecting party. For example, when updating the current operating system (e.g. Windows Updates) the user wants to be sure that the updates are taken from the genuine vendor site. If the updates are not authentic, they can be malicious, and the security of the system can be put under a serious threat.

- *Non-repudiation* - prevention of the party from denying previously taken actions. Once the party has agreed on something, e.g. has signed a contract, later, it cannot deny it. If brought to court, a judge can easily decide if the party has taken the controversial action.

Cryptology can be divided into two main areas: cryptography and cryptanalysis. The cryptography is the part that deals with construction and proposal of new cryptographic algorithms, primitives and protocols as well as implementation of these object. Cryptanalysis, on the other hand, examines and evaluates the security of the cryptographic objects, i.e. it points weaknesses and strengths in the analyzed objects and presents methods to improve the security. Cryptography and cryptanalysis are tightly related and often it is not clear the border where the first ends and the second starts. Moreover, they complement each other – it is virtually impossible to propose a new primitive (i.e. block cipher or hash function) without giving a preliminary security analysis of the primitive.

It is critical to understand that the security of a cryptographic primitive is evaluated in a framework where the adversary is an intelligent enemy. For example, it is a general misunderstanding, that codes are cryptographic objects[4]. Yet, basically they protect the data only against the laws of nature (some bits of the communicated data are randomly flipped, erased or added). On the other hand, an adversary can easily undermine the security of a code when used for enciphering, by altering the "right" rather than random bits. Therefore, when proposing a new primitive, the cryptographer should evaluate the security against an intelligent enemy – a human. The primitive has to withstand all the known attacks as well as possible new attacks.

Cryptography is further divided into symmetric (private-key) and asymmetric (public-key) cryptography. In symmetric cryptography both trusted parties share the same secret key - the first party enciphers the data with the secret key, and the second party deciphers it with the same key. In asymmetric cryptography each party has two keys, a public key which is known to everyone and used to encipher data and a private key which only the party knows and which is used to decipher the data. Hence, everyone can send a message to the party since the public key is known, but only the party can read the message since the private key is available only to the party.

There are four main areas of symmetric cryptography:

- Cryptographic hash functions

[4]This misunderstanding comes from the fact that through history indeed codes were used for confidentiality, i.e. to encipher information.

- Block ciphers
- Stream ciphers
- Message authentication codes

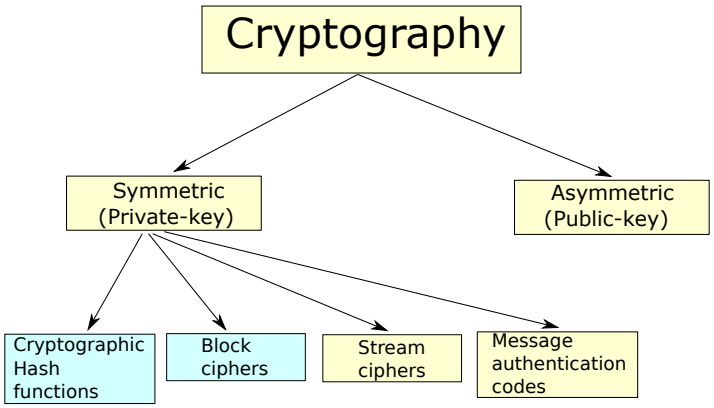

Figure 1: The division of cryptography. This thesis deals with cryptographic hash function and block ciphers.

This work is devoted to the cryptanalysis of the first two types of symmetric primitives, i.e. cryptographic hash functions and block ciphers (see Fig. I). The thesis is structured as follows. In part I we give the definitions and the security requirements of hash functions and block ciphers. Part II introduces some of the basic techniques of cryptanalysis, in particular, the techniques used in the further attacks. The contribution is presented in parts III, IV, and V.

Chapter 1

Cryptographic Hash Functions

Cryptographic hash functions are one of the main four symmetric cryptography primitives. The notion of a cryptographic hash function is tightly related to the notion of hash function which does not have any security (cryptographic) properties. A hash function takes as an input a message of arbitrary length, and produces an output of fixed length called message digest, hash value, or hash. The process of producing this output is called hashing. Formally, n-bit hash function $h(M)$ is defined as:

$$h(M) : \{0,1\}^* \rightarrow \{0,1\}^n.$$

In order to be cryptographic, a hash function is required to have three security properties:

1. *Collision resistance* - it is hard to find two distinct messages that hash to the same value, i.e. it is computationally infeasible to find M_1, M_2 such that $h(M_1) = h(M_2)$.

2. *Preimage resistance* - it is hard to find a message that produces some target hash value, i.e. given H^*, it is computationally infeasible to find M such that $h(M) = H^*$.

3. *Second-preimage resistance* - it is hard to find another (second) message that produces the same hash as the first message, i.e. given message M, it is computationally infeasible to find M' such that $h(M') = h(M)$.

Any hash function that satisfies the above three properties can be considered to be cryptographic. However, in some applications, only a single property is required. For example, in some protocols it is sufficient the hash function to be only preimage resistant[1]. In our further work, we assume that a cryptographic hash function has all three security properties. Several additional properties such as resistance against length extension attacks, absence of large input-output correlations, are usually expected as well from a hash function. We will use the terms hash function

[1] This type of functions are called non-invertible or one-way functions.

and cryptographic hash function interchangeably to denote collision, preimage and second-preimage resistant hash function.

The three security properties, along with the fixed-length output property, dictate the areas of application of cryptographic hash functions:

- *Digital signatures* are used to provide authentication of data (a message, file or document). They are the digital equivalents of the conventional handwritten signatures. The public-key cryptography provides algorithms for digitally signing data. However, these algorithms are generally considered to be very slow (several magnitudes slower than private-key algorithms) hence when applied to the whole message (which can be as long as several megabytes), they lead to a rather inefficient algorithms. Instead of signing the whole message, the user can first produce a hash value of the message and then sign only the hash of the message. Since the hash is short, the efficiency of the public-key algorithms becomes irrelevant. The collision resistance property of the hash function ensures that a potentially malicious user cannot easily produce two messages that hash to the same value. Otherwise, he can argue that he has signed the malicious message (instead of the original).

- *Data integrity* can be achieved using hash functions. The sending party sends the whole message over a possibly insecure channel, and the hash of the message over a secure channel. The receiving party gets the message, hashes it, and checks if the message digest is identical to the one received. If the hashes are the same, then the data is unaltered. The second-preimage resistance of a hash function guarantees that an adversary cannot easily create a second malicious message that has the same hash as the original, and hence he cannot alter the original message.

- *Password storing* is another important area of application of hash functions. Instead of storing the original user passwords, a computer system can store only the hashes of the passwords [2]. When a user types his password, the system compares the hash of the typed password with the stored hash of the original password. If they match, the user is allowed to log in. Note that with this solution, the file with the hashes of the original passwords has to be only write protected (but not read protected). The preimage resistance of the hash function ensures that a malicious user, even though he has the file with the hashes, cannot find a password that has some of the hash value from the file.

So far, we have omitted the exact definition of the terms "hard" and "computationally infeasible" used in the description of the security properties. The only parameter of a cryptographic hash function is the length n of the output. Therefore, the notion of "hardness" is defined as a function of n.

Definition 1 *An n-bit hash function $h(M)$ is collisions resistant if the complexity, measured in the number of invocations of $h(M)$, of finding collisions is $2^{\frac{n}{2}}$.*

[2] Unix-type systems use this type of password storing.

Definition 2 *An n-bit hash function $h(M)$ is preimage resistant if the complexity, measured in the number of invocations of $h(M)$, of finding preimage is 2^n.*

Definition 3 *An n-bit hash function $h(M)$ is second preimage resistant if the complexity, measured in the number of invocations of $h(M)$, of finding second preimage is 2^n.*

The estimates $2^{\frac{n}{2}}, 2^n, 2^n$ have a theoretical basis. There exist generic attack algorithms, i.e. algorithms applicable to any hash function, that find collisions and (second) preimages with these workloads.

The generic collision finding algorithm is based on the so-called birthday paradox which states that in a room of 23 people the probability that two of them have a birthday on the same day is higher than $1/2$. Assuming that a year has 365 days, this probability can be found as:

$$1 - \frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{343}{365} \approx 0.507. \tag{1.1}$$

This means that if the output has 365 different elements then it is sufficient to pick 23 random inputs to get a collision with a probability higher than $1/2$. Let us try to find the same number of inputs when the output has N elements, i.e. let us try to find k such that:

$$1 - \frac{N}{N} \cdot \frac{N-1}{N} \cdot \frac{N-2}{N} \cdot \ldots \cdot \frac{N-(k-1)}{N} = \frac{1}{2}.$$

Note that $\frac{N-s}{N} = 1 - s/N$, and $1 - 1/N \approx e^{-1/N}$. Hence (1.1) can be rewritten as:

$$e^{-1/N} e^{-2/N} \ldots e^{-(k-1)/N} = 1/2. \tag{1.2}$$

With a further reduction, we obtain:

$$e^{-k(k-1)/(2N)} = 1/2. \tag{1.3}$$

Since k is a large number, we can replace $k-1$ with k. This way we obtain the solution for (1.3):

$$k \approx 1.18\sqrt{N}. \tag{1.4}$$

For an n-bit hash function the value of N is 2^n, and therefore a collision can be found after $1.18\sqrt{2^n} \approx 2^{n/2}$ hash function invocations.

The generic (second) preimage attack works as follows. The attacker sequentially produces k hashes of randomly taken input messages. The output space has 2^n distinct elements, hence the probability p that some of hash values of these k inputs coincide with the target value H^* is given by the formula:

$$p = 1 - \left(\frac{2^n - 1}{2^n}\right)^k. \tag{1.5}$$

Again, we use the asymptotic approximation $(1 + 1/n)^n = e$, and obtain

$$p = 1 - e^{-\frac{k}{2^n}}. \tag{1.6}$$

Therefore, when $k = 2^n$, with probability $p = 0.63$ one can find (second) preimage for an n-bit hash function.

In a hash function, although the input message can be of an arbitrary length, the output hash value always has a fixed length. To cope with this, hash functions are built iteratively from smaller primitives called compression functions. The most popular design that explores this idea is the Merkle-Damgård design [100, 45]. A compression function f takes as an input a chaining value H^{old} and a message block M and produces as an output another chaining value H^{new}. Given f, one can build a hash function $h(M)$ as follows (See Fig. 1):

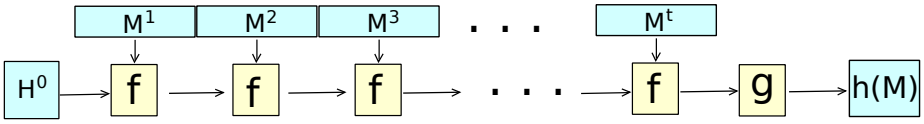

Figure 1.1: Merkle-Damgård Construction

1. Concatenate to the input message M the length of the message[3], i.e. produce $\tilde{M} = M||length(M)$.

2. Split the message $\tilde{M}$ into blocks of equal size, i.e. $\tilde{M} = M^1||M^2||\ldots||M^t$, where $|M^i| = m$.

3. Fix some initial chaining value H^0. Produce consecutively

$$H^i = f(H^{i-1}, M^i), i = 1, \ldots, t.$$

4. The hash of M is $g(H^t)$, where g is some output transformation[4], i.e.

$$h(M) = g(H^t).$$

Conventionally, the security of a cryptographic hash function is analyzed in a framework where the function is regarded as a whole, indivisible object. However, a more conservative approach has become a standard in the analysis of functions in the last 15 years. Rather than the whole hash function, the main target of attack became the underlying compression function. Some of the attack scenarios on the compression functions seem highly unlikely to have any effect on the security of the whole hash function, which afterwards is the final goal. However, it is a common opinion in the cryptographic community that even a slight weakness in the compression function may be explored and later may lead to a weakness in the hash function.

[3] If necessary, add padding to make the length multiple of m.

[4] Usually, the output transformation is an identity function, a truncation, or some simple non-cryptographic mapping.

1.1 Collision Attacks

Collision attacks explore weaknesses in the function that lead to producing collisions faster than $2^{\frac{n}{2}}$ function invocations. The advancement of cryptanalysis increased the number of types of collision attacks on both the hash function as well as on the compression function. First, let us focus on attacks on the latter (See [86]). Depending on the freedom to choose the input parameters, the following types of collision attacks on compression functions can be launched:

- Real collisions - given a chaining value H, find two distinct messages M_1, M_2 such that $f(H, M_1) = f(H, M_2)$.
- Pseudo-collisions - find two distinct chaining values H_1, H_2 and a message M such that $f(H_1, M) = f(H_2, M)$.
- Near collisions - find a chaining value H and two distinct messages M_1, M_2 such that $H_w(f(H, M_1) \oplus f(H, M_2)) << n$, where H_W is the Hamming weight function.
- Semi-free start collisions - find a chaining value H and two distinct messages M_1, M_2 such that $f(H, M_1) = f(H, M_2)$.
- Free start collisions - find two distinct chaining values H_1, H_2 and two distinct messages M_1, M_2 such that $f(H_1, M_1) = f(H_2, M_2)$.

A collision attack on the compression function can be as well a combination of some of the above attacks, e.g. semi-free start near collision.

When the output size is n bits, generic attacks can produce the above collisions with complexity of $2^{\frac{n}{2}}$ compression function invocations. For near collisions, when the Hamming weight is at most w, i.e. when the output chaining values of the messages coincide in at least $n - w$ specific bits, near collisions can be produced generically with an effort of $2^{\frac{n-w}{2}}$ invocations[5].

Only a real collision attack on the compression functions directly leads to a collision attack on the hash functions. For this purpose, the attacker finds real collisions for the compression function with a fixed input chaining value to the initial chaining value of the hash function. At the first glance, the importance of the rest of the collision attacks on the compression functions seems questionable. However, it is often a practice among the cryptologists to gradually improve the attack on some function to the point when it becomes a real collisions attack. In other words, first some simpler attack on the compression function is found, e.g. pseudo-collisions. Then using more advanced techniques, this attack is extended to a real collision attack. Also, some of the collision attacks on the hash functions are actually a combination of various collision attacks on the compression function. For example, the collision attack on the hash standard SHA-1 [140] is a combination of a near collision attack and a semi-free start collision attack.

[5]This estimate is correct when the position of the coincided bits is fixed. Otherwise, the complexity drops to $2^{n/2} \cdot \sqrt{C_n^w}$.

1.2 Preimage and Second Preimage Attacks

In the (second) preimage attacks the adversary finds (second) preimages for an n-bit function with less than 2^n invocations. Similarly as in the collision attacks, in the preimage attacks the adversary can find preimages for the whole hash function or only for the compression function. In the framework of compression functions, the following attacks are interesting:

- Real (second) preimage attack - given chaining values H, H^*, find a message M such that $f(H, M) = H^*$.
- Pseudo-(second) preimage attack - given a chaining value H^*, find a message M and a chaining value H such that $f(H, M) = H^*$.

Obviously, a real (second) preimage attack on the compression function can be converted to a (second) preimage attack on the hash function by simply fixing the input chaining value H to the initial chaining value H^0. This assumes that the output transformation g is easily invertible. If this is not the case, the attacker can still launch a second preimage attack. For that matter, instead for the final hash value, he finds a real second preimage for the chaining value that is input to g (see Fig. 1).

When the size of the intermediate chaining value is same as the size of the hash value, so-called single pipe hash functions, the pseudo preimage attack on the compression function can be converted to a preimage attack on the hash function [86]. First the attacker finds a lot of pseudo preimages for H^*, i.e. he builds a set $S_2 = \{(H_2, M_2) | f(H_2, M_2) = H^*\}$. Then, by taking random messages, he constructs a lot of intermediate chaining values from the initial chaining value, i.e. he builds a set $S_1 = \{(H_1, M_1) | f(H^0, M_1) = H_1\}$. When $|S_1| \cdot |S_2| \geq 2^n$ then there is a high probability that these two sets intersect in the first element of the pair, i.e. $H_1 = H_2$. Then the preimage for H^* is the message $M_1 || M_2$.

1.3 Distinguishers

Modern hash function are often used as building blocks for other cryptographic primitives. The compression functions alone are sometimes misused for purposes other than the one they were created for, e.g. by removing the feed-forward some compression functions can be turned into block ciphers. Therefore, a new type of security requirements is imposed on hash and compression functions. Besides the traditional collision and (second) preimage resistance, a hash and its underlying compression function are supposed to behave as a random oracle[6] – a function that outputs a random value for each input. In this framework, any property that might distinguish the function from the random oracle is considered to be a security threat for the function. However, there are properties that hold for any function, but not for the oracle simply because the oracle does not have a finite definition.

[6] For a full definition of a random oracle see [14].

It is interesting to note that an exact definition of a valid distinguisher on hash/compression functions is still missing. This is mostly due to the fact that for each function there exist some trivial distinguishers that are applicable since the attacker knows (and can control) all the input parameters of the function and the function is public. Intuitively, a distinguisher is valid if it is not trivial.

Chapter 2

Block Ciphers

Block ciphers are the oldest cryptographic primitives. Their main purpose is to provide confidentiality, i.e. securely exchange messages and data over an insecure channel.

A block cipher $E(K, P)$ takes as an input an n-bit string P called a plaintext and k-bit string K called a key, and produces a n-bit string C called a ciphertext:

$$E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n.$$

When the key $K \in \{0,1\}^k$ is fixed, the block cipher $E_K(P) \equiv E(K, P)$ becomes a permutation on $\{0,1\}^n$. The process of producing the ciphertext from the plaintext and the key is called encryption. The reverse process, i.e. producing the plaintext from the ciphertext and the key, is called decryption.

To communicate over an insecure channel (see Fig. 2), the two trusted parties first fix a secret key K known only to them. Then the first party encrypts messages with the block cipher using the key K and sends the ciphertext of the messages to the second party. This party decrypts the ciphertext with the key K and obtains the initial messages.

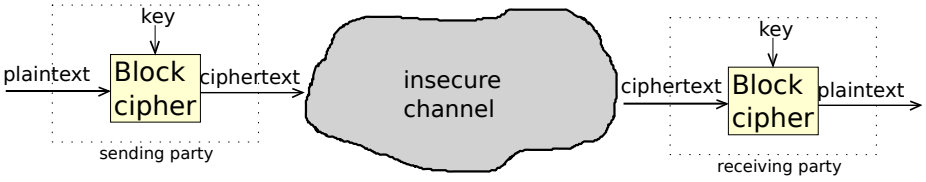

Figure 2.1: Secure communication with a block cipher

If a malicious user (i.e. an adversary) intercepts this communication (the ciphertexts), he cannot produce the initial messages because he does not have the key used for decryption. Hence, as long as the key is kept secret and the block cipher is secure, the confidentiality is preserved.

2.1 Distinguishers and Key-recovery Attacks

A block cipher is secure if an adversary, who does not know the secret key but has a lot of pairs plaintext-ciphertext, cannot obtain any additional information about the block cipher or the secret key from these pairs. When the adversary does obtain information about the block cipher then we say he has constructed *a distinguisher* for the cipher. If he can obtain information about the key, which is indeed his ultimate goal, then he has launched *a key-recovery attack* on the cipher.

Depending on the attack framework, the adversary can have access to various types of plaintext-ciphertext pairs. In the weakest (but most practical) case, the adversary has a set of only ciphertexts (without the corresponding plaintexts). This is *the ciphertext-only* attack. When he has an access to both the plaintexts and the corresponding ciphertexts then it is called *a known-plaintext* attack. To test the robustness of a block ciphers a few more assumptions were introduced. Although the attacks in these frameworks are highly unrealistic, the resistance of a cipher to them provides a good sign of security. These are *the chosen-plaintext* attack, where the adversary can choose the plaintexts and obtain the corresponding ciphertexts, and *the chosen-ciphertext* attack, where he can choose the ciphertexts and get the plaintexts. In the chosen-type of attacks, if the adversary can choose the next ciphertext (or plaintext) that he wants to decrypt (or encrypt) depending on the previously obtained pairs, then it is *an adaptive* attack. If he selects all the ciphertexts (plaintexts) and obtain the corresponding pairs of plaintexts (ciphertext) all at once, then it is *a non-adaptive* attack.

Now, let us focus on distinguishers for a cipher E. The main task of the adversary in this type of attacks is to show that he can determine, without knowing the key, if some cipher is the target E. More specifically, the adversary is given so-called black box access either to the cipher E with some fixed, but unknown to the adversary key or to some random permutation. The adversary can query this black box, i.e. he can fix plaintexts and obtain the corresponding ciphertexts and vice-versa. After he collects a number of plaintext-ciphertext pairs he is supposed to decide, with a sufficiently high probability, if the black box is the target cipher E. In general, to find a distinguisher for E the adversary provides some property for the cipher that holds for all the keys. Then he collects an amount of plaintext-ciphertext pairs to confirm the existence of the property. The adversary knows what is the amount of these pairs sufficient to confirm the property. Hence, after collecting this critical amount (from the black box), if the property has been confirmed, then the black box is the target cipher E, otherwise it is some random permutation. The adversary should keep in mind that if the critical amount of pairs is high that even a random permutation can exhibit this property. Therefore, to be successful he has to find a property for the cipher which can be confirmed with sufficiently low number of plaintext-ciphertext pairs.

In the key recovery attacks the adversary tries to get information about the key, i.e. he tries to find some or all bits of the key. For a cipher E with a k-bit key, given only a few pairs of plaintext-ciphertext the adversary can always check all possible 2^k keys, and thus find all k bits. This is the so-called brute force of the key space – it is a generic attack applicable to any cipher and requires around 2^k encryptions.

To launch a key-recovery attack the adversary has to discover a method of finding the key bits faster than brute force. He can target as well a partial-key recovery, when he tries to find k_0 bits ($k_0 < k$) of the key. Then, the total complexity of the attack should be lower than 2^{k_0}.

2.1.1 Related-key Attacks and Classes of Weak Keys

In the distinguishers and the key-recovery attacks it is assumed that there is one single key, fixed randomly and unknown to the adversary. In the *related-key* attacks this requirement is relaxed, and instead of providing black-box access to the cipher with a fixed unknown single random key K, the adversary is given access to the cipher with several fixed unknown keys, but with a known relation between them. More precisely, the adversary has an access to the ciphers $E_{K_1}(P), E_{K_2}(P), \ldots, E_{K_s}(P)$, where K_1 is some fixed unknown key, $K_i = f_i(K_1), i = 2, \ldots, s$, and the relations f_i are chosen by the adversary. The adversary, however, can choose specific relations f_i which would allow him to trivially launch key-recovery attacks. Therefore, some relations are not permitted. One very commonly used relation is XOR or modular addition, i.e. $s = 2$, and $K_2 = f_2(K_1) = K_1 \oplus \Delta$ (or $f_2(K_1) = K_1 + \Delta$).

The attack frameworks of distinguishers and key recovery assume that the key is chosen randomly from the set $\tilde{K}$ of all possible keys (in the related-key attacks the first key is chosen randomly). In some cases, the adversary can find attacks that are applicable to the cipher only if the key is randomly chosen from some subset $S \subset \tilde{K}$. The subset S is called *a class of weak keys*. Let $|\tilde{K}| = 2^k$ and $|S| = 2^s$. To have a valid attack based on the class of weak keys S, the adversary has to take into account the probability that a randomly chosen key belongs to S, which is 2^{s-k}. Hence, the complexity of attacks for classes of weak keys has always an additional factor of 2^{k-s} which is the workload required to get a key from the weak class.

2.2 Open-key Distinguishers

In the *open-key distinguishers* for block ciphers the adversary knows or chooses the key for the cipher E. These type of distinguishers are relatively new and were first introduced in the work of Knudsen and Rijmen [85]. They have presented attacks on ciphers assuming that the adversary knows the key – these type of distinguisher are *the known-key distinguishers*. Later, Biryukov, Khovratovich and Nikolić [29] have shown attacks on ciphers when the adversary can even choose the key used for encryption – these are *the chosen-key distinguishers*.

Unlike the traditional distinguishers for block ciphers, where the adversary does not know the key, in the open-key distinguishers the adversary has the key. Therefore, the open-key distinguishers are based on the same principe as the distinguishers for hash functions and a formal definition of valid open-key distinguishers is still missing.

Part II

Some Techniques for Cryptanalysis of Hash Functions and Block Ciphers

The field of cryptanalysis is constantly growing and new methods and techniques enrich the arsenal of cryptanalysis every once in a while. Listing and describing all these techniques would require a lot of space and effort, and at the end some will still be missing. Therefore, we will describe only the techniques that are used in our attacks as well as the new techniques that appeared in the last few years. This selection does not include some of the very powerful and widely used approaches of cryptanalysis. To learn more about these methods, an interested reader is referred to linear cryptanalysis [92], impossible cryptanalysis [18], high-order differentials [83], slide attacks [34], multiset attacks [41, 33] and others.

Chapter 3

Differential Analysis

Differential analysis is one of the most applied and successful technique for analysis of cryptographic hash functions and block ciphers. It was officially introduced by Biham and Shamir in 1990 for analysis of DES [21], although indications exist that the technique was known before to some government agencies such as the U.S. National Security Agency and IBM.

The idea of differential analysis is to encrypt a pair of plaintexts with a specific input difference between them and to check if the corresponding pair of ciphertexts has some specific output difference. If the probability, that a randomly chosen plaintext pair with the fixed input difference produces a ciphertext pair with a fixed output difference, is higher than a mere chance, then this leads to a distinguisher that in some cases can be converted to a key-recovery attack.

Formally, differential analysis can be introduced as follows. Let $E_K(P)$ be some n-bit block cipher with a key size of k bits. Let Δ_I, Δ_O be the input and the output differences in the plaintexts, and in the corresponding ciphertexts, i.e.

$$\begin{aligned} \Delta_I &= P_1 \oplus P_2, \\ \Delta_O &= E_K(P_1) \oplus E_K(P_2). \end{aligned}$$

A differential $\Delta = (\Delta_I, \Delta_O)$ is a pair of input difference Δ_I and output difference Δ_O. The probability P_Δ of the differential Δ is defined as:

$$P_\Delta \equiv P(E_K(P) \oplus E_K(P \oplus \Delta_I) = \Delta_O)$$

Note that for a randomly chosen Δ_I, Δ_O the value of P_Δ is around 2^{-n}. On the other hand, if $P_\Delta > 2^{-k}$ then this differential, denoted as $\Delta_I \xrightarrow{P} \Delta_O$, leads to a valid distinguisher for the block cipher $E_K(P)$.

Although at first glance, launching a differential attack seems a trivial task, the real problem lies in finding good input and output differences. This problem is usually reduced to the problem of finding *differential characteristics* (or trails). Note that a differential only fixes the values of the differences in the plaintext (Δ_I) and in the ciphertext (Δ_O). One can also fix the values of the differences in some

intermediate states, e.g. after each round of the cipher, or even after each transformation in the cipher. A differential characteristic $\tilde{\Delta} = (\Delta_I, \Delta_1, \Delta_2, \ldots, \Delta_t, \Delta_O)$ is a set of differences in the plaintext, in some t intermediate states, and in the ciphertext. Similarly, as in the case of differential, for characteristic we can introduce probabilities p_i that a fixed difference Δ_i after some round (or transformation) will evolve to Δ_{i+1}. Then a characteristic $\tilde{\Delta}$ can be denoted as:

$$\Delta_I \xrightarrow{p_1} \Delta_1 \xrightarrow{p_2} \Delta_2 \xrightarrow{p_3} \ldots \xrightarrow{p_t} \Delta_t \xrightarrow{p_{t+1}} \Delta_O$$

The probability of the characteristic is $P_{\tilde{\Delta}} = \prod p_i$.

A convenient method of constructing a characteristic for a large number of rounds for the cipher, was described as well in the work of Biham and Shamir [21]. This is the notion of iterative characteristic – it is on a small number of rounds of the cipher, but it has the same input and output difference. To produce a long characteristic, the attacker only has to concatenate many such short iterative characteristics. He can do so, since the input and the output differences of the characteristic are the same. Therefore, if the iterative characteristic is on k rounds and it has a probability p, then the attacker can construct a characteristic on $t \cdot k$ rounds with a probability p^t.

A differential can be seen as a collection of characteristics that have common input and output differences (Δ_I and Δ_O). This way, the probability of the differential increases with each new characteristic found. In practice this rarely happens[1], i.e. the attacker usually finds and uses only one differential characteristic. Having a differential $\Delta_I \xrightarrow{p} \Delta_O$ (or a characteristic) with high probability, the attacker can easily construct a so-called differential distinguisher. He encrypts around $1/p$ pairs of plaintexts $(P^i, P^i \oplus \Delta_I), i = 1, \ldots, p$ and produces the corresponding pair of ciphertexts (C_1^i, C_2^i). Then, with high probability he can expect that one of the ciphertext pair has the difference Δ_O, i.e. for some $j, C_1^j \oplus C_2^j = \Delta_O$. In a random permutation the pair of plaintexts/ciphertexts with Δ_I, Δ_O differences can be found after around 2^n encryptions. Hence, when $p \gg 2^{-n}$, the attacker has a distinguisher for the cipher.

Although we have defined differential analysis for block ciphers, the notion of differentials and differential characteristics can easily be transferred to cryptographic hash function. The main application of differential trails to hash functions is for collision search – a trail for the function that ends with a zero difference (i.e. $\Delta_O = 0$) leads to a collision attack. If Δ_O has low Hamming weight then this can lead to a near collision. On the other hand any trail (with sufficiently high probability) can be used as differential distinguisher for the function.

3.1 Message Modification for Differential Attacks

In some attack frameworks, e.g. hash (compression) function attacks or chosen-key attacks on block ciphers, the adversary has the freedom to choose all input param-

[1] In ciphers with diffusion layer based on MDS matrices, e.g. AES, many characteristics can be found that compose a round-reduced differential.

eters. Usually, this permits him to fix the values of one or more intermediate states. This has a significant impact on differential attacks. The differential characteristics for some hash function/block cipher specify a set of differences in some intermediate states and probabilities that differences will propagate. These probabilities are evaluated under the assumption that the values of the intermediate states are random, i.e. in the differential pair, the first state has some random values S, while the second state has a value $S \oplus \Delta_I$. However, when the attacker controls all the inputs, usually he can control the exact values of a number of intermediate states. Hence, using the message freedom the attacker can fix the required value S for the state such that the differences will propagate with probability 1 through this round. In cryptanalysis, the authors refer to the message modification technique when they use the freedom of the input (not necessarily only the message) to fix some intermediate states which leads to differential attacks with lower complexities.

3.2 Truncated Differentials

Truncated differentials, introduced by Knudsen [83], are differential-type of attacks that are usually applied to byte-oriented primitives, i.e. the primitives that use transformations on bytes (rather than on words or bits). Knudsen noted that when constructing differential characteristics for these types of primitives, instead of specifying the exact values of the differences for each separate byte, one can only indicate if there is a difference in the byte or not – the bytes with differences are called active bytes, while the bytes without difference are called non-active bytes.

The advantages of considering this type of differentials depend on the actual underlying transformations used in the primitive. Most of the byte-oriented primitives are substitution-permutation (SP) networks, i.e. in each round they have a substitution layer which is a byte-wise application of non-linear functions, called S-boxes, and a permutation layer which is usually some linear transformation of the bytes of the state. The S-box layer is used to destroy the linearity[2], while the permutation layer is used to introduce diffusion among the bytes, i.e. after a low number of rounds, each byte of the state depends on all bytes of the input. In general, for SP primitives it is much easier to build a truncated than a regular characteristic. When the S-boxes are bijective (which usually is the case for most of the SP primitives) then after the application of the S-boxes each active byte stays active and vice-versa with probability 1. Hence, to build a truncated characteristic the attacker only has to focus on the permutation layer in each round of the primitive and only in this layer the probability of the characteristic can be lower than 1. Usually, the probability of a truncated trail is different for different direction (backward or forward). An example of a 2-round truncated differential characteristic for AES is given in Fig. 3.1.

[2]Otherwise the whole primitive becomes a linear transformation and it can easily be distinguished.

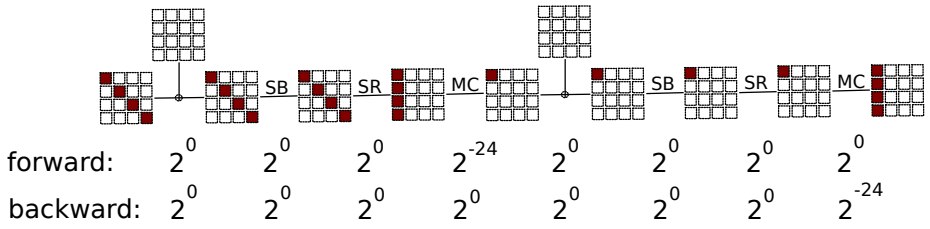

Figure 3.1: Truncated trail for two rounds of AES with the probabilities in forward and backward direction

3.2.1 Rebound Attack

The rebound attack [98] was introduced to enhance the efficiency of truncated differentials[3]. It can be used in contents where the attacker controls the value of the state. Therefore, this technique is used in various collision/differential attacks on hash functions and open-key distinguishers on block ciphers.

As mentioned before, the probability of truncated trails can be less than 1 only in the permutation layer of the primitive. One can easily apply message modification and pass one permutation layer for free (with probability 1). The rebound attack, on the other hand, is used to pass two consecutive permutation layers for free. The attacker starts with some difference in the state at the input of the permutation layer in the first round, and a difference in the state at the output of the permutation layer in the second round. He goes forward in the first permutation layer and produces the difference Δ_I in the state at the end of the first round. Similarly, he goes backwards through the second permutation layer and obtains the difference Δ_O at the input of this layer. Since between these two layers lies only on S-box layer, the attacker now has to match these two differences through the S-box layer, i.e. to find a value X such that the differential pair $(X, X \oplus \Delta_I)$ at the beginning of the second round satisfies $(Sbox(X) \oplus Sbox(X \oplus \Delta_I) = \Delta_O$. It is important to notice that for any value X the trail in the permutation layers in the two rounds does not change because these layers are linear. The matching of the differences can be done byte-wise since the S-box layer is byte-oriented. Once the attacker has fixed the proper value of X, the value of the states becomes fixed as well hence the attacker can easily produce the values of the initial and final states and check if the difference in the states is as predicted by the trail. In Fig. 3.2 we give a rebound attack on 7 rounds of AES [95].

3.2.2 Super S-boxes

The idea of Super S-boxes [58, 87] is closely related to the idea of the rebound attack. This technique is applicable to primitives that have incomplete one round

[3]Recently, the rebound attacks has been applied as well to enhance other, non-differential attacks (See [78]).

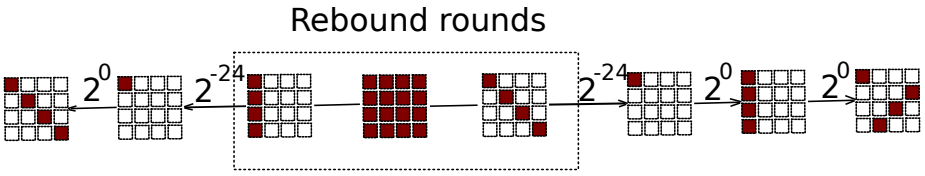

Figure 3.2: Truncated trail for 7 rounds of AES with the Rebound attack in the middle two rounds.

diffusion. Depending on the diffusion of the primitive, Super S-boxes can fix good values in three or even more rounds of the truncated trail. For this purpose, the attacker first divides 1.5 rounds of the cipher into independent parts – he can do so since the diffusion is incomplete. Then, as in the rebound attack, he tries to match the differences but now, instead of going 1-round-forward and 1-round-backward, he can use 1.5-round-forward and 1.5-round-backward and instead of byte-wise matching, he uses bigger S-boxes (e.g. word-oriented).

3.3 Local Collisions

The technique of local collisions was first used by Chabaud and Joux in the collision attack on the hash function SHA-0 [38]. It is a method of building efficiently differential characteristic that can be used for collisions. The idea is to introduce a difference in the state of the function at some round through the injected message word, and then to cancel the further full-state expansion of this difference by introducing differences in the message words of the following few rounds. After the last local collision round, the state is free of differences.

In SHA-0, Chabaud and Joux considered a 6-round local collision, where in the first round they put a difference in a single bit of the message word, and in the next 5 rounds, differences in various bits of the message words. To build the characteristic on the full hash function, they used local collisions starting at different rounds. Note that is not possible to use only one local collision because of the message expansion in SHA-0 – each message word is used a number of times through the rounds of the function.

The local collision attack technique was applied for the first time to block ciphers in the recent attacks on the encryption standard AES [28, 29]. There, the subkeys played the role of message words with differences. The local collisions for AES consist of only two rounds.

3.4 Boomerang Attacks

After the invention of differential analysis, the designers of cryptographic primitives put special attention on proving the resistance of the primitives against differential

attacks. This was usually done by computing the upper bound on the probability of the best trail for the cipher. If the upper bound is p, then the complexity of producing a differential pair is around $1/p$, hence when $1/p > 2^n$, producing the pair requires more effort than a simple brute force and thus the trail cannot be used for an attack. Wagner [139] showed that this method is not sufficient to prove the resistance against differential attacks, and the designer also has to show that not only the probability of the trail for the whole cipher has to be low, but also the probability of round-reduced trails.

Let E be the analyzed cipher, and let E_0, E_1 be two subciphers of E such that $E = E_1 \circ E_0$. Let $\Delta \to \Delta^*$ be some differential trail for E_0 that holds with probability p and $\nabla \to \nabla^*$ be a trail for E_1 with probability q. An attacker starts with a pair of plaintexts $(P_1, P_2) = (P_1, P_1 \oplus \Delta)$ and produces a pair of corresponding ciphertexts $(C_1, C_2) = (E(P_1), E(P_2))$. By XORing ∇^* to the pair, he obtains a new pair of ciphertext $(C_3, C_4) = (C_1 \oplus \nabla^*, C_2 \oplus \nabla^*)$, decrypts this pair, and gets the corresponding pair of plaintexts $(P_3, P_4) = (E^{-1}(C_3), E^{-1}(C_4))$ (see Fig 3.3). Then

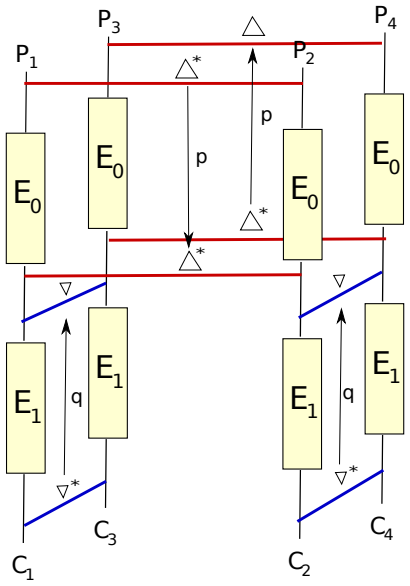

Figure 3.3: The boomerang attack on cipher $E = E_1 \circ E_0$

the difference $P_3 \oplus P_4$ is Δ with probability at least p^2q^2 since:

1. the difference $E_0(P_1) \oplus E_0(P_2)$ is Δ^* with probability p;
2. the differences $E_1^{-1}(C_1) \oplus E_1^{-1}(C_3)$, $E_1^{-1}(C_2) \oplus E_1^{-1}(C_4)$ are both ∇ with probability q^2;
3. when 1), 2) hold, then the difference $E_1^{-1}(C_3) \oplus E_1^{-1}(C_4)$ is Δ^* (with probability pq^2) and $E^{-1}(C_3) \oplus E^{-1}(C_4)$ is Δ with probability p^2q^2.

The quartet of plaintexts (P_1, P_2, P_3, P_4) and corresponding ciphertexts (C_1, C_2, C_3, C_4) such that $P_1 \oplus P_2 = P_3 \oplus P_4 = \Delta$ and $C_1 \oplus C_3 = C_2 \oplus C_4 = \nabla^*$ is called a boomerang quartet. Producing such quartet for a random permutation requires around 2^n, hence when $p^2q^2 > 2^{-n}$, the attacker has a distinguisher for the cipher.

Chapter 4

Preimage Attacks

The (second) preimage resistance of a cryptographic hash function plays a crucial role in many applications of the function. Most of the so far published (second) preimage attacks on hash functions, are purely theoretical, i.e. the complexities of the attacks are lower than 2^n function calls, but much higher than some practical complexities in the range of up to 2^{60} calls.

A common attack technique for finding preimages is the meet-in-the-middle technique and its advanced variant splice-and-cut technique.

4.1 Meet-in-the-middle attacks

The meet-in-the-middle attacks (MITM) are used to find the values of some of the inputs of a cryptographic primitive, when the rest of the inputs and the output is fixed. Let $G(x, y)$ be some cryptographic primitive (a compression function or a block cipher). The MITM is used to solve the problem:

$$G(x, A) = B, \tag{4.1}$$

i.e. to find an input x when the input y is fixed to A and the output is fixed to B.

The MITM can be applied only if G can be represented as a composition of two primitives G_1, G_2, each having as an input some independent (of the other primitive) bits of x, and G_2 is invertible. Let $x = (x_1, x_2, x_3)$, where x_1 (x_2) are the bits of x that are inputs only to G_1 (G_2), and x_3 are inputs to both G_1 and G_2. Then (4.1) can be rewritten as:

$$G_2(x_2, x_3, G_1(x_1, x_3, A)) = B. \tag{4.2}$$

If we take into account the invertible property of G_2, we get:

$$G_1(x_1, x_3, A) = G_2^{-1}(x_2, x_3, B). \tag{4.3}$$

Therefore, to find the solution x of (4.1) we have to find a value C such that $C = G_1(x_1, x_3, A) = G_2^{-1}(x_2, x_3, B)$, i.e. we have to search for a collision between the

functions $G_1(x_1, x_3, A)$ and $G_2^{-1}(x_2, x_3, B)$. This search can trivially be performed as follows (see Fig. 4.1). Let the intermediate value C have n bits. First we fix x_3 to some arbitrary value K. Then we generate sets S_1 and S_2 each as $2^{n/2}$ evaluations of $G_1(x_1, K, A)$ and $G_2^{-1}(x_2, K, B)$ at some arbitrary random points x_1 and x_2. Since $|S_1| \cdot |S_2| = 2^{n/2} \cdot 2^{n/2} = 2^n$, with probability close to $1/2$ the sets S_1 and S_2 have a common element, which is the searched value C.

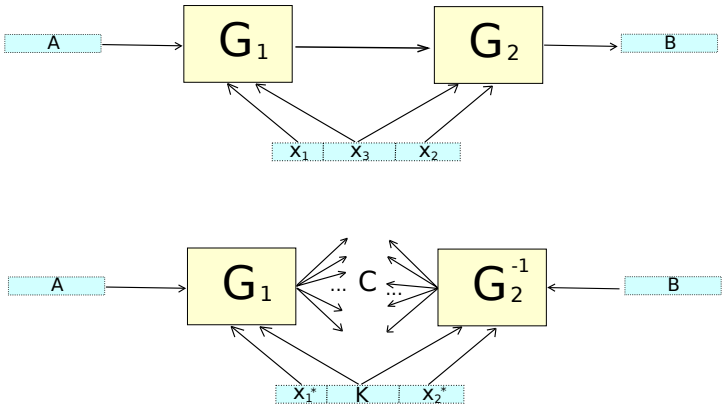

Figure 4.1: The meet-in-the-middle attack

4.2 Splice-and-cut technique

The splice-and-cut technique [4], proposed by Aoki and Sasaki, has been successfully applied to find preimages for various cryptographic hash function [123, 124, 125, 126, 1, 127]. Although this approach has been applied only to hash functions with compression functions based on block ciphers in the Davies-Meyer mode, i.e. $f(H, M) = E_M(H) \oplus H$, it can equally well be applied to functions based on some other modes as well.

Splice-and-cut is built upon the meet-in-the-middle approach. As mentioned above, one of the requirements of the MITM is the inversion property of G_2. Since $f(H, M)$ is based on the Davies-Meyer mode, any decomposition of $f(H, M)$ into two functions would lead to the second function being non-invertible because of the feedforward of H. The splice-and-cut technique divides the r round compression function $f(H, M)$ into three consecutive round segments: the first r_1 rounds, the next r_2 rounds and the last r_3 rounds (obviously $r_1 + r_2 + r_3 = r$). The exact values of r_i are chosen such that the first and the third segment on one side and the second segment on the other side have some unique independent input message words (notice the similarity with the MITM requirements). Let M be the input message and $M = (m_1, m_2, m_3)$, where m_1 is being input only to the first and/or

the third segment, m_2 only to the second segment, and m_3 to any. Let x be the value of the internal state of $f(H,M)$ after the first r_1 rounds (at the beginning of the second round segment), and y be the value after the first $r_1 + r_2$ rounds (at the beginning of the third round segment). Let $f_1(H, m_1, m_3)$ be the function of the first r_1 rounds, $f_1(x, m_2, m_3)$ of the next r_2 rounds, $f_3(y, m_1, m_3)$ of the last r_3 rounds, and H^* be the target preimage value. Then

$$x = f_1(H, m_1, m_3) \tag{4.4}$$

$$y = f_2(x, m_2, m_3) \tag{4.5}$$

$$H^* = f_3(y, m_1, m_3) \oplus H. \tag{4.6}$$

First we fix any values $\tilde{m_3}, \tilde{y}$ for m_3, and y. For the value of H from (4.6) we get that $H = f_3(\tilde{y}, m_1, \tilde{m_3}) \oplus H^*$. Therefore, from (4.4) and (4.5) we get:

$$f_1(f_3(\tilde{y}, m_1, \tilde{m_3}) \oplus H^*, m_1, \tilde{m_3}) = x \tag{4.7}$$

$$f_2^{-1}(\tilde{y}, m_2, \tilde{m_3}) = x. \tag{4.8}$$

To find a preimage for H^* we only have to find a collision in x. Note that each equation has an independent input – m_1 in the first and m_2 in the second. Therefore, we can construct again the sets S_1 and S_2 as in the MITM attack and get the required preimage.

Chapter 5

Various Distinguishers

The most widely used distinguishers are the differential distinguishers. However, most of the new hash function/block cipher designs are built to stop these type of attacks, i.e. after a reduced number of rounds the success probability of the differential distinguishers becomes negligible. Therefore, the cryptographic community has found other types of distinguishers. Among them are the rotational and cube distinguishers.

5.1 Rotational Analysis

Rotational distinguishers exploit the fact that some transformations produce rotated outputs for rotated inputs. Let F be a transformation and $(X, X \lll_r)$ be a pair of inputs for F, where $\lll_r$ is a cyclic rotation to the left by r bits. This pair is called rotational if

$$F(X) \lll_r = F(X \lll_r).$$

For this pair we say that F preserves the rotational property. The input and the output of a transform can be a single word or a vector of words, i.e. $\tilde{X} = (X_1, \ldots, X_n)$. Then, a rotational input/output pair is defined as $(\tilde{X}, \tilde{Y})$, where $Y_i = X_i \lll_r$, and $i = 1, \ldots, n$. A system $\Phi(\tilde{X})$ that consists of the transformations $F_1, \ldots, F_k$ preserves the rotational property if it produces a rotational output pair for a rotational input pair.

Two important issues have to be addressed. The first one – unlike in the differential analysis, where the adversary may introduce differences only in some part of the input, in the rotational analysis, all the input pairs of words have to be rotational. The second issue – there are only a few transformations that preserve the rotational property for any input pair. In the majority of cases, for an arbitrary input X, the condition $F(X) \lll_r = F(X \lll_r)$ holds with a probability p_F. This probability is further called a rotational probability of F and it depends on the rotation amount r. If we assume that the outputs of the transformations are independent, a system Φ composed of transformations $F_1, \ldots, F_k$ preserves the rotational property with probability $p_\Phi = p_{F_1} \cdot p_{F_2} \cdot \ldots \cdot p_{F_k}$. Hence, in order to find the probability that a

system preserves the rotational property, one only has to find the probabilities that each instance of the underlying transformations preserves this property. For a random system with n-bit output, the probability that a rotational input will produce a rotational output is 2^{-n}. Therefore, if a system Φ with n-bit output, has a rotational probability $p_\Phi > 2^{-n}$, then this system can be distinguished from a random system.

5.2 Cube Attacks and Cube Testers

A cube attacks [49] is an algebraic method of cryptanalysis targeting block ciphers. When applicable, it can lead to a key recovery. The main idea of cube attacks is to find linear terms in the algebraic normal form of the output. If these terms are bits of the secret key, then the attacker can easily solve the linear system and thus recover the key bits.

The method works in two phases. In the first, so-called preprocessing phase, given the description or a black box access to the cipher, the attacker tries to build the algebraic normal form (ANF) for the output bits (the input variables $x_1, \ldots, x_n$ are the bits of the key and the plaintext). However, it is reasonable to assume that the explicit formula of the ANF, i.e. the number of monomials in the ANF, is exponential in the number of input bits, hence the attacker cannot fully construct ANF. Given a monomial t_I that is a product of variables with indices from I, where $I \subset \{1, \ldots, n\}$, the ANF of some output bit can be represented as:

$$p(x_1, \ldots, x_n) = t_I \cdot p_{S(I)} + q(x_1, \ldots, x_n), \tag{5.1}$$

where $p_{S(I)}$ is called a superpoly of I in p and has no common variables with t_I, and q misses at least one variable from t_I. The attacker, using various heuristics, tries to find different t_I's that have linear non-constant superpolies – these type of terms are called maxterms of p. Note that if in (5.1) the attacker sums over all possible values of the variables of t_I, it can be shown that the right part of (5.1) becomes equal to $p_{S(I)}$, while the value of the left part is known to the attacker. Hence, when enough maxterms are available, the problem of recovering the key bits (the one in $p_{S(I)}$) is reduced to the problem of solving a system of linear equations. This is done in the online phase, when the key is fixed and unknown to the attacker. He queries the cipher with plaintexts in order to obtain the sum over all possible values of variables of different maxterms (found previously in the preprocessing phase). Once he gets the bits of the ciphertexts, he only has to solve the linear system of equations to recover the key bits included in the maxterms.

Cube testers [6] are distinguishing type of attacks that further exploit some properties of superpolies. As in the cube attacks, the adversary chooses various t_I's and gets their superpolies $p_{S(I)}$. Then he tries to show some distinguishing property of these superpolies, i.e. the attacker tries to show that these superpolies have some property that is not (easily) found in superpolies of a random function/permutation. Usually such properties are balance, constantness, low degree and other. Cube testers are purely practical type of distinguisher, i.e. the attacker can present the distinguishing property in feasible time and finding testers with higher complexity is still an open problem.

Part III

Differential Attacks on Hash Functions

Differential attacks play an important role in the analysis of cryptographic hash function. The starting point of the attacks is finding a high probability differential trail for the hash function or the underlying compression function. The search for these trails is usually ad-hoc, i.e. there is no universal method applicable to any function. Depending on the properties of the found trail, the attacker can launch several distinct differential attacks:

1. When the differential trail for n-bit function ends with a zero difference, and the probability of the trail is higher than $2^{-\frac{n}{2}}$, then the attacker can launch a simple collision attack. Given a trail $\Delta_I \xrightarrow{2^{-t}} 0$ for an n-bit function $F(x)$ the attacker composes 2^t pairs $(F(X_i), F(X_i \oplus \Delta_I)$. Then with a high probability he can expect that in one of the pairs he will get a collision. Since $t < n/2$, he finds this collision with an effort less than $2^{\frac{n}{2}}$, i.e. he has launched a collision attack. Further, we present trails and collision attacks for the compression functions of SHA-256 and LAKE. The attacks were published in:

   - [109] **Collisions for Step-Reduced SHA-256**, FSE 2008
   - [27] **Cryptanalysis of the LAKE Hash Family**, FSE 2009

2. When the trail has a probability higher than 2^{-n} (but not necessarily higher than $2^{-\frac{n}{2}}$), then it can be used as a distinguisher for the function. Again the attacker creates 2^t pairs and finds one that follows the trail. On the other hand, in a random function, he needs around 2^n pairs to find one pair that on the fixed input differences produces the fixed output difference. It is important to notice that the estimate 2^n for a random function holds only when the input difference is fixed as well – otherwise the complexity drops to $2^{\frac{n}{2}}$.

3. Two high probability trails on different halves of the function can be used in a boomerang-type of attack. When the combined probability of these two trails is higher than $2^{-\frac{n}{2}}$, then the attacker can create a boomerang distinguisher (a boomerang quartet) with a complexity lower than 2^n, which is the complexity in the generic case. Further, we present the details for the boomerang distinguisher on the SHA-3 proposal BLAKE. The attack is based on the work:

   - [32] **Boomerang Attacks on BLAKE-32**, FSE 2011

Chapter 6

Collisions for SHA-2

The SHA-2 family of hash functions was introduced to the cryptographic community as a new, more complex, and hopefully, more secure variant of the MD4-family of hash functions. The recent results on the widely used MD4-family hash functions SHA-1 and MD5 [140],[141] show flaws in the security of these functions, with respect to collision attacks. The question arises, if the most complex member of MD4-family, the SHA-2 family, is also vulnerable to collision attacks.

Research has been made on finding local collisions for the SHA-2 family. Gilbert and Handschuh [56] reported a 9-step local collision with probability of the differential path of 2^{-66}. Later, Mendel et al. [97] estimated the probability of this local collision to be 2^{-39}. Somitra and Palash obtained a local collision with probability 2^{-42}. Using modular differences Hawkes, Paddon and Rose [64] were able to find a local collision with probability 2^{-39}. Finding a real collision for SHA-2 was due to Mendel et al [97]. They studied the message expansion of the SHA-256 and reported a 19-step near collision.

We find a 9-step differential trail (we use modular differences) that holds with probability $\frac{1}{3}$ if we fix some of the intermediate values and solve the equations that arise, i.e. if we use a message modification. We show that it is not necessary to introduce differences in message words on each step of the trail. This helps us to overcome the message expansion. Using only one instance of this differential trails we find 20 and 21-step collisions (collisions for the original initial value) with complexities of 3 and 2^{19} compression function calls, respectively. Also, using slightly different differential paths we are able to find a 23-step semi-free start collision (collisions for a specific initial value) with a complexity of 2^{21} calls. Our final result is a 25-step semi-free start near collision with Hamming distance of 15 bits and 2^{34} calls.

Our results were further improved by Indesteege et al [66]. They reported 24-step collisions for SHA-256 and SHA-512 with complexity of $2^{28.5}$ and 2^{53} compression function calls, respectively. They were able to find as well free-start near-collisions for 31-step reduced SHA-256.

6.1 Description of SHA-2

SHA-2 family consists of iterative hash functions SHA-224, SHA-256, SHA-384, and SHA-512. For our purposes, we will describe only SHA-256. The definitions of the rest of the functions can be found in [136]. The SHA-256 takes a message of length less than 2^{64} and produces a 256-bit hash value. First, the input message is padded so the length becomes a multiple of 512, and afterwards each 512-bit message block is processed as an input in the Damgård-Merkle iterative structure. Each iteration calls a compression function which takes for an input a 256-bit chaining value and a 512-bit message block and produces an output 256-bit chaining value. The output chaining value of the previous iteration is an input chaining value for the following iteration. The initial chaining value, i.e. the value for the first iteration, is fixed, and the chaining value produced after the last message block is proceeded, is the hash value of the whole message. Internal state of SHA-256 compression function consists of eight 32-bit variables A, B, C, D, E, F, G, and H, each of which is updated on every of the 64 steps. These variables are updated according to the following equations:

$$\begin{aligned}
A_{i+1} &= \Sigma_0(A_i) + Maj(A_i, B_i, C_i) + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + H_i + K_i + W_i \\
B_{i+1} &= A_i \\
C_{i+1} &= B_i \\
D_{i+1} &= C_i \\
E_{i+1} &= \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + H_i + K_i + W_i + D_i \\
F_{i+1} &= E_i \\
G_{i+1} &= F_i \\
H_{i+1} &= G_i.
\end{aligned}$$

The $Maj(X,Y,Z)$ and $Ch(X,Y,Z)$ are bitwise boolean functions defined as:

$$\begin{aligned}
Ch(X,Y,Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\
Maj(X,Y,Z) &= (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z).
\end{aligned}$$

For SHA-256 $\Sigma_0(X)$ and $\Sigma_1(X)$ are defined as:

$$\begin{aligned}
\Sigma_0(X) &= ROTR^2(X) \oplus ROTR^{13}(X) \oplus ROTR^{22}(X) \\
\Sigma_1(X) &= ROTR^6(X) \oplus ROTR^{11}(X) \oplus ROTR^{25}(X).
\end{aligned}$$

State update function uses constants K_i, which are different for every step. The 512-bit message block itself is divided in 16 32-bit words: $m_0, m_1, \ldots, m_{16}$. Afterwards, the message block is expanded to 64 32-bit words according to the following rule:

$$W_i = \begin{cases} m_i, & 0 \leq i \leq 15 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}, & i > 15 \end{cases}$$

For SHA-256 $\sigma_0(X)$ and $\sigma_1(X)$ are defined as:

$$\begin{aligned}
\sigma_0(X) &= \quad ROTR^7(X) \oplus ROTR^{18}(X) \oplus SHR^3(X) \\
\sigma_1(X) &= ROTR^{17}(X) \oplus ROTR^{19}(X) \oplus SHR^{10}(X).
\end{aligned}$$

Table 6.1: A 9 step differential trail for the SHA-2 family. Notice that only 5 differences are introduced, i.e. in the steps $i, i+1, i+2, i+3$, and $i+8$.

step	ΔA	ΔB	ΔC	ΔD	ΔE	ΔF	ΔG	ΔH	ΔW
i	0	0	0	0	0	0	0	0	1
i+1	1	0	0	0	1	0	0	0	δ_1
i+2	0	1	0	0	-1	1	0	0	δ_2
i+3	0	0	1	0	0	-1	1	0	δ_3
i+4	0	0	0	1	0	0	-1	1	0
i+5	0	0	0	0	1	0	0	-1	0
i+6	0	0	0	0	0	1	0	0	0
i+7	0	0	0	0	0	0	1	0	0
i+8	0	0	0	0	0	0	0	1	δ_4
i+9	0	0	0	0	0	0	0	0	0

The compression function after the 64-th step adds the initial values to the chaining variables, i.e. the hash result of the compression function is:

$$h(M) = (A_{64}+A_0, B_{64}+B_0, C_{64}+C_0, D_{64}+D_0, E_{64}+E_0, F_{64}+F_0, G_{64}+G_0, H_{64}+H_0).$$

These values become the initial chaining value for the next compression function.

6.2 Technique for Creating Collisions

Differences used in our analysis are subtractions mod 2^{32} differences.
We use the following notation:

$$\Delta X = X^{'} - X,\ X \in \{A, B, D, D, E, F, G, H, W, m\},$$
$$\Delta Maj^i(\Delta a, \Delta b, \Delta c) = Maj(A_i + \Delta a, B_i + \Delta b, C_i + \Delta c) - Maj(A_i, B_i, C_i),$$
$$\Delta Ch^i(\Delta e, \Delta f, \Delta g) = Ch(E_i + \Delta e, F_i + \Delta f, G_i + \Delta g) - Ch(E_i, F_i, G_i).$$
$$\Delta\Sigma_0(A_i) = \Sigma_0(A_i^{'}) - \Sigma_0(A_i)$$
$$\Delta\Sigma_1(E_i) = \Sigma_1(E_i^{'}) - \Sigma_1(E_i)$$
$$\Delta\sigma_0(m_i) = \sigma_0(m_i^{'}) - \sigma_0(m_i)$$
$$\Delta\sigma_1(m_i) = \sigma_1(m_i^{'}) - \sigma_1(m_i)$$

We introduce perturbation (i.e. difference in the message word) on step i and in the following 8 steps we try to cancel the differences in the internal variables. We use the differential trail presented in Tbl. 6.1. As you can see from the table (column ΔW), only the perturbation (the initial input difference of the local collisions) has been fixed to 1. All the other differences are to be determined.

6.2.1 Conditions for the Local Collision

From the definition of SHA-2, focusing on registers A_{i+1} and E_{i+1}, we get:

$$\Delta A_{i+1} - \Delta E_{i+1} = \Delta\Sigma_0(A_i) + \Delta Maj^i(\Delta A_i, \Delta B_i, \Delta C_i) - \Delta D_i,$$
$$\Delta E_{i+1} = \Delta\Sigma_1(E_i) + \Delta Ch^i(\Delta E_i, \Delta F_i, \Delta G_i) + \Delta H_i + \Delta D_i + \Delta W_i.$$

We keep in mind that when $\Delta A_i = \Delta B_i = \Delta C_i = 0$ then $\Delta Maj^i(0,0,0) = 0$. Also when $\Delta E_i = \Delta F_i = \Delta G_i = 0$ then $\Delta Ch^i(0,0,0) = 0$.

We fix the differences for the registers A and E (as shown in Tbl. 6.1). The variables B, C, D, F, G, H can only inherit the values from A and E. So, for each step we get some equations with respect to δ_i and A_i or E_i.

Step i+1. We have that $\Delta D_i = 0$, $\Delta H_i = 0$, $\Delta\Sigma_0(A_i) = 0$, $\Delta\Sigma_1(E_i) = 0$. We require $\Delta A_{i+1} = 1$, $\Delta E_{i+1} = 1$. So we deduce:

$$\Delta W_i = 1 \tag{6.1}$$

Step i+2. We have that $\Delta D_{i+1} = 0$, $\Delta H_{i+1} = 0$. We require $\Delta A_{i+2} = 0$, $\Delta E_{i+2} = -1$. We want also $\Delta\Sigma_0(A_{i+1}) = 1$ to be satisfied. So we deduce:

$$\Delta Maj^{i+1}(1,0,0) = 0, \tag{6.2}$$
$$\Delta W_{i+1} = -1 - \Delta Ch^{i+1}(1,0,0) - \Delta\Sigma_1(E_{i+1}). \tag{6.3}$$
$$\Delta\Sigma_0(A_{i+1}) = 1 \tag{6.4}$$

Step i+3. We have that $\Delta D_{i+2} = 0$, $\Delta H_{i+2} = 0$, $\Delta\Sigma_0(A_{i+2}) = 0$. We require $\Delta A_{i+3} = 0$, $\Delta E_{i+3} = 0$. So we deduce:

$$\Delta Maj^{i+2}(0,1,0) = 0, \tag{6.5}$$
$$\Delta W_{i+2} = -\Delta\Sigma_1(E_{i+2}) - \Delta Ch^{i+2}(-1,1,0). \tag{6.6}$$

Step i+4. We have that $\Delta D_{i+3} = 0$, $\Delta H_{i+3} = 0$, $\Delta\Sigma_0(A_{i+3}) = 0$, $\Delta\Sigma_1(E_{i+3}) = 0$. We require $\Delta A_{i+4} = 0$, $\Delta E_{i+4} = 0$. So we deduce:

$$\Delta Maj^{i+3}(0,0,1) = 0, \tag{6.7}$$
$$\Delta W_{i+3} = -\Delta Ch^{i+3}(0,-1,1). \tag{6.8}$$

Step i+5. We have that $\Delta D_{i+4} = 1$, $\Delta H_{i+4} = 1$, $\Delta\Sigma_0(A_{i+4}) = 0$, $\Delta\Sigma_1(E_{i+4}) = 0$. We require $\Delta A_{i+5} = 0$, $\Delta E_{i+5} = 1$. So we deduce:

$$\Delta Ch^{i+4}(0,0,-1) = -1. \tag{6.9}$$

Step i+6. We have that $\Delta D_{i+5} = 0$, $\Delta H_{i+5} = -1$, $\Delta\Sigma_0(A_{i+5}) = 0$. We require $\Delta A_{i+6} = 0$, $\Delta E_{i+6} = 0$. We want also $\Delta\Sigma_0(E_{i+5}) = 1$ to be satisfied. So we deduce:

$$\Delta Ch^{i+5}(1,0,0) = 0. \tag{6.10}$$
$$\Delta\Sigma_1(E_{i+5}) = 1 \tag{6.11}$$

Step i+7. We have that $\Delta D_{i+6} = 0$, $\Delta H_{i+6} = 0$, $\Delta\Sigma_0(A_{i+6}) = 0$, $\Delta\Sigma_1(E_{i+6}) = 0$. We require $\Delta A_{i+7} = 0$, $\Delta E_{i+7} = 0$. So we deduce:

$$\Delta Ch^{i+6}(0,1,0) = 0. \tag{6.12}$$

Step i+8. We have that $\Delta D_{i+7} = 0$, $\Delta H_{i+7} = 0$, $\Delta\Sigma_0(A_{i+7}) = 0$, $\Delta\Sigma_1(E_{i+7}) = 0$. We require $\Delta A_{i+8} = 0$, $\Delta E_{i+8} = 0$. So we deduce:

$$\Delta Ch^{i+7}(0,0,1) = 0. \tag{6.13}$$

Step i+9. We have that $\Delta D_{i+8} = 0$, $\Delta H_{i+8} = 1$, $\Delta\Sigma_0(A_{i+8}) = 0$, $\Delta\Sigma_1(E_{i+8}) = 0$. We require $\Delta A_{i+9} = 0$, $\Delta E_{i+9} = 0$. So we deduce:

$$\Delta W_{i+8} = -1. \tag{6.14}$$

6.2.2 Solution of the System of Equations

Let us first observe (6.4) and (6.11). From the differential trail we can see that $\Delta A_{i+1} = \Delta E_{i+5} = 1$. It means that we want the functions $\Delta\Sigma_0(A_{i+1}), \Delta\Sigma_1(E_{i+5})$ to preserve the difference 1, in other words:

$$\Sigma_0(A_{i+1} + 1) - \Sigma_0(A_{i+1}) = 1,$$
$$\Sigma_1(E_{i+5} + 1) - \Sigma_1(E_{i+5}) = 1.$$

The only solution to these equations is $A_{i+1} = E_{i+5} = -1$, so we get:

$$A_{i+1} = -1, \qquad A^{'}_{i+1} = 0, \tag{6.15}$$
$$E_{i+5} = -1, \qquad E^{'}_{i+5} = 0. \tag{6.16}$$

Now let us consider the function $\Delta Maj^i = Maj(A^{'}_i, B^{'}_i, C^{'}_i) - Maj(A_i, B_i, C_i)$. Let us suppose that $B^{'}_i = B_i$, $C^{'}_i = C_i$ and A_i and $A^{'}_i$ differ in every single bit, i.e. $A_i \oplus A^{'}_i =$`0xffffffff`. Then:

$$\Delta Maj^i = 0 \Leftrightarrow B_i = C_i$$

Therefore (6.2) gives us $B_{i+1} = C_{i+1}$, which is $A_i = A_{i-1}$. With the same reasoning we can deduce from (6.5) that $A_{i+2} = A_i$, and from (6.7) that $A_{i+3} = A_{i+2}$. Hence, from (6.2),(6.5) and (6.7) we get:

$$A_{i-1} = A_i = A_{i+2} = A_{i+3} \tag{6.17}$$

Similarly to what we have done with Maj, now let us consider ΔCh^i and suppose that $F^{'}_i = F_i$, $G^{'}_i = G_i$ and E_i and $E^{'}_i$ differ in every single bit. Then:

$$\Delta Ch^i = 0 \Leftrightarrow F_i = G_i$$

Therefore (6.10) and (6.16) gives us $F_{i+5} = G_{i+5}$, which is:

$$E_{i+4} = E_{i+3} \tag{6.18}$$

Solving (6.12) requires slightly different reasoning; if we have $E_{i+6} = E'_{i+6}$, $G_{i+6} = G'_{i+6}$ and F_{i+6} and F'_{i+6} would differ in every bit (and they do, see (6.16)) then :

$$\Delta Ch^{i+6} = 0 \Leftrightarrow E_{i+6} = 0. \tag{6.19}$$

Analogously, from (6.13) we get:

$$E_{i+7} = -1 \tag{6.20}$$

The only remaining condition is (6.9):

$$\Delta Ch^{i+4} = Ch(E_{i+4}, F_{i+4}, G'_{i+4}) - Ch(E_{i+4}, F_{i+4}, G_{i+4}) = -1,\ G'_{i+4} - G_{i+4} = -1.$$

The words $E_{i+4}, F_{i+4}, G_{i+4}$ are already determined to satisfy the previous conditions. Therefore, we do not have any degrees of freedom left to control precisely the solution of this equation. Therefore we will try to find the probability that this condition holds. We can see that it holds if and only if register E_{i+4} has 0's in the bits where G'_{i+4} and G_{i+4} are different. The G'_{i+4} and G_{i+4} can differ in the last i bits, where $1 \leq i \leq 32$., and these bits are uniquely determined. So, for the probability we get:

$$\sum_{i=1}^{i=32} \text{P\{Last i bits of } E_{i+4} \text{ are zero\}} \times \text{ P\{Difference in exactly i last bits\}} =$$

$$= \sum_{i=1}^{i=32} \frac{1}{2^i} \frac{1}{2^i} \approx \frac{1}{3}.$$

So, the overall probability of our differential trail (when certain conditions are satisfied) is $\frac{1}{3} = 2^{-1.58}$.
The differences in the message words of the differential as in Tbl. 6.1 are the following:

$$\begin{aligned}
\delta_1 &= -1 - \Delta Ch^{i+1}(1,0,0) - \Delta\Sigma_1(E_{i+1}),\\
\delta_2 &= -\Delta\Sigma_1(E_{i+2}) - \Delta Ch^{i+2}(-1,1,0),\\
\delta_3 &= -\Delta Ch^{i+3}(0,-1,1)\\
\delta_4 &= -1
\end{aligned}$$

Notice that the condition (6.17) shows us that A_i=B_i has to hold.

6.3 Full, Semi-free and Near Collisions for Step-reduced SHA-256

Our attack technique is the following:

1. Introduce perturbation (difference) at step i;

2. Correct the differences in the following 8 steps (probability of success is the probability of our differential trail, i.e. $\frac{1}{3}$). After the last step of the differential trail (local collision), the differences in the internal variables are zero;

3. All the message words injected after the last step of the local collision have to have zero differences;

6.3.1 20-Step Collision

In Tbl. 6.2 we can see that the words m_5, m_6, m_7, m_8, and m_{13} are used only once in the first 20 steps of SHA-2, i.e. they are not used to compute the values of expanded words W_{16}, W_{17}, W_{18}, and W_{19}. This means that the message expansion does not introduce any difference after the last step of the differential trail. Hence, we get collision for 20 step reduced SHA-2, and these collisions can be found practically by hand. The complexity of finding a collision is $2^{1.58}$ compression function calls.

6.3.2 21-Step Collision

From Tbl. 6.2 we can easily see that we have to consider the message expansion since there are no message words that are used only once in the first 21 steps and that have the proper indexes for the differential trail.
We will introduce differences in the words m_6, m_7, m_8, m_9, and m_{14}. The words m_6, m_7, m_8 are used only once in the first 21 steps. Therefore the message expansion in the first 21 steps is irrelevant with respect to these words, i.e. differences in these words do not introduce any other new differences, after the last step of the differential (step 14). Now, we want to find words $m_9, m_9', m_{14}, m_{14}'$ such that after the 14-th step, the message expansion will not introduce any difference in the following steps. From Tbl. 6.2 we see that the words m_9 and m_{14} are used in W_{16}, W_{18}, and W_{20}. So, from the definition of W_i we get the equations:

$$\Delta W_{16} = \Delta\sigma_1(m_{14}) + \Delta m_9 + \Delta\sigma_0(m_1) + \Delta m_0 = 0 \tag{6.21}$$
$$\Delta W_{17} = \Delta\sigma_1(m_{15}) + \Delta m_{10} + \Delta\sigma_0(m_2) + \Delta m_1 = 0 \tag{6.22}$$
$$\Delta W_{18} = \Delta\sigma_1(W_{16}) + \Delta m_{11} + \Delta\sigma_0(m_3) + \Delta m_2 = 0 \tag{6.23}$$
$$\Delta W_{19} = \Delta\sigma_1(W_{17}) + \Delta m_{12} + \Delta\sigma_0(m_4) + \Delta m_3 = 0 \tag{6.24}$$
$$\Delta W_{20} = \Delta\sigma_1(W_{18}) + \Delta m_{13} + \Delta\sigma_0(m_5) + \Delta m_4 = 0 \tag{6.25}$$

Obviously if $m_i' = m_i$ ($W_i' = W_i$) then $\Delta\sigma_0(m_i) = 0$ ($\Delta\sigma_0(W_i) = 0$). This means that $\Delta W_{17} = \Delta W_{19} = 0$. If we can make so that $\Delta W_{16} = 0$ then $\Delta W_{18} = \Delta W_{20} = 0$. Therefore, we get the equation:

$$\Delta\sigma_1(m_{14}) + \Delta m_9 = 0 \tag{6.26}$$

Considering that $\Delta m_{14} = \delta_4 = -1$, and m_9 can take any value, our experimental results (Monte Carlo method with 2^{32} trials) give us a probability of $2^{-17.5}$ that Δm_{14} and Δm_9 satisfy this equation. Therefore, the overall probability of the differential trail used for the 21-step collision is around 2^{-19}.

Table 6.2: Message expansion of SHA-2. There is an 'x' in the intersection of row with index i and column with index j if W_i uses m_j.

W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	x															
1		x														
2			x													
3				x												
4					x											
5						x										
6							x									
7								x								
8									x							
9										x						
10											x					
11												x				
12													x			
13														x		
14															x	
15																x
16	x	x								x					x	
17		x	x								x					x
18	x	x	x	x						x		x			x	
19		x	x	x	x						x		x			x
20	x	x	x	x	x	x				x		x		x	x	
21		x	x	x	x	x	x				x		x		x	x
22	x	x	x	x	x	x	x	x		x		x		x	x	x

6.3.3 23-Step Semi-free Start Collision

For 23 step collision we introduce differences in the words m_9, m_{10}, m_{11}, and m_{12}. If we want to follow our differential trail, then we are supposed to introduce difference in the message word W_{17}. We cannot control W_{17} directly because it is an expanded word. From the condition $W_{17} = \delta_4 = -1$ (differential) and the message expansion, we get:

$$\Delta W_{17} = \Delta\sigma_1(m_{15}) + \Delta m_{10} + \Delta\sigma_0(m_2) + \Delta m_1 = -1.$$

Since $\Delta m_{15} = \Delta m_2 = \Delta m_1 = 0$, we get:

$$\Delta m_{10} = -1. \tag{6.27}$$

In our original differential trail there are no message differences in the word W_{16}. But for W_{16} we have:

$$\Delta W_{16} = \Delta\sigma_1(m_{14}) + \Delta m_9 + \Delta\sigma_0(m_1) + \Delta m_0.$$

Obviously only $\Delta m_9 \neq 0$ and therefore $\Delta W_{16} = \Delta m_9 = 1 \neq 0$. Therefore we shall use slightly different differential trail – one with a difference in the word W_{16}. To keep everything else intact, the equations for the step 17 become the following:

$$\Delta E_{17} = \Delta\Sigma_1(E_{16}) + \Delta Ch^{16}(0,0,1) + \Delta D_{16} + \Delta H_{16} + \Delta W_{16}.$$

From the differential we can see that: $\Delta E_{17} = \Delta\Sigma_1(E_{16}) = \Delta D_{16} = \Delta H_{16} = 0$. Therefore we get:

$$\Delta Ch^{16}(0,0,1) + \Delta W_{16} = 0. \tag{6.28}$$

Now, let us observe the other words of the message expansion.
For W_{18} we have:

$$W_{18} = \Delta\sigma_1(W_{16}) + \Delta m_{11} + \Delta\sigma_0(m_3) + \Delta m_2 = 0$$

Since $\Delta m_3 = \Delta m_2 = 0, \Delta W_{16} = 1$ we get the equation:

$$\Delta\sigma_1(W_{16}) + \Delta m_{11} = 0. \tag{6.29}$$

For W_{19} we have:

$$W_{19} = \Delta\sigma_1(W_{17}) + \Delta m_{12} + \Delta\sigma_0(m_4) + \Delta m_3 = 0$$

Since $\Delta m_4 = \Delta m_3 = 0, \Delta W_{17} = -1$ we get the equation:

$$\Delta\sigma_1(W_{17}) + \Delta m_{12} = 0. \tag{6.30}$$

For W_{20} we have:

$$W_{20} = \Delta\sigma_1(W_{18}) + \Delta m_{13} + \Delta\sigma_0(m_5) + \Delta m_4 = 0$$

Since $\Delta W_{18} = \Delta m_{13} = \Delta m_5 = \Delta m_4 = 0$ we get that this equation is satisfied for all values of W_{18}, m_{13}, m_5, m_4.
For W_{21} we have:

$$W_{21} = \Delta\sigma_1(W_{19}) + \Delta m_{14} + \Delta\sigma_0(m_6) + \Delta m_5 = 0$$

Since $\Delta W_{19} = \Delta m_{14} = \Delta m_6 = \Delta m_5 = 0$ we get that this equation is satisfied for all values of W_{19}, m_{14}, m_6, m_5.
For W_{22} we have:

$$W_{22} = \Delta\sigma_1(W_{20}) + \Delta m_{15} + \Delta\sigma_0(m_7) + \Delta m_6 = 0$$

Since $\Delta W_{20} = \Delta m_{15} = \Delta m_7 = \Delta m_6 = 0$ we get that this equation is satisfied for all values of W_{20}, m_{15}, m_7, m_6.
For W_{23} we have:

$$W_{23} = \Delta\sigma_1(W_{21}) + \Delta W_{16} + \Delta\sigma_0(m_8) + \Delta m_7 = 0$$

Since $\Delta W_{21} = \Delta m_8 = \Delta m_7$ and $\Delta W_{16} \neq 0$ we get that this equation has no solution. That is why we can not get more than 23 step collision.

Let us try to solve (6.27), (6.28), (6.29) and (6.30).
For (6.27) and the value of the register E_{11} from the trails' conditions we have:

$$\Delta E_{11} = \Delta\Sigma_1(E_{10})) + \Delta Ch^{10}(1,0,0) + \Delta m_{10}.$$

Since $\Delta E_{11} = m_{10} = -1$ we get:

$$\Delta\Sigma_1(E_{10}) + \Delta Ch^{10}(1,0,0) = 0.$$

We solve this equation by setting $\Delta\Sigma_1(E_{10}) = 1$ and $\Delta Ch^{10}(1,0,0) = -1$. The first one has solution:

$$E_{10} = -1,\ E_{10}^{'} = 0. \tag{6.31}$$

The second equation holds for the values:

$$F_{10} = G_{10} + 1. \tag{6.32}$$

Now let us turn to the solution of (6.28). Using the fact that $G_{16} = -1$ and $G_{16}^{'} = 0$, we get that this equation is satisfied if:

$$E_{16} = \texttt{0xfffffffe} \tag{6.33}$$

Let us observe the equation (6.30). From the conditions of the differential trail we have:

$$\Delta E_{13} = \Delta\Sigma_1(E_{12}) + \Delta Ch^{12}(0,-1,1) + \Delta H_{12} + \Delta D_{12} + \Delta m_{12}$$

Since $\Delta E_{13} = \Delta E_{12} = \Delta H_{12} = \Delta D_{12} = 0$ we get:

$$\Delta Ch^{12}(0,-1,1) + \Delta m_{12} = 0.$$

If we substitute m_{12} from (6.30) we can get:

$$\Delta Ch^{12}(0,-1,1) = \Delta\sigma_1(-1).$$

This equation can be satisfied if we can control E_{12} and F_{12}.
For E_{12}, from the definition of A_{12} and E_{12} we have:

$$A_{12} - E_{12} = \Sigma_1(A_{11}) + Ch(A_{11}, B_{11}, C_{11}) - D_{11}$$

Considering that $A_{12} = A_{11} = C_{11} = D_{11}$ from the trails's conditions, we get:

$$E_{12} = A_9 - \Sigma_1(A_9)$$

Since A_9 can take any value (we consider semi-free start collision) we deduce that E_{12} can take any value.
The F_{12} value, which is E_{11} can be controlled through H_{10}. Notice that changing H_{10}, which is G_9, does not effect E_{10}, because from (6.31) we can see that E_{10} always takes the arranged value.
We proved that we can fully control E_{12} and F_{12}. We can choose some specific value for $\Delta\sigma_1(-1)$ which is possible to get from $\Delta Ch^{12}(0,-1,1)$, and set the A_9 and G_9 so that the equation (6.30) will hold.
The last equation, i.e. (6.29), is satisfied for some specific values of W_{16} and m_{11}. Our experimental results show that with probability $2^{-19.5}$ W_{16} and m_{11} satisfy (6.29). Therefore the overall probability of semi-free start collision for 23-step reduced SHA-256 is around 2^{-21}.

6.3.4 25-Steps Semi-free Start Near Collision

Let us assume that we have a semi-free start collision on the 23-rd step. Each following step introduces differences in the chaining variables A and E. The variables B, C, D, F, G, H can only inherit differences from A and E. Therefore, for each step, we should try to minimize the differences in A and E. When we say to minimize the differences we mean to minimize the Hamming distances between $A^{'}$ and A, and between $E^{'}$ and E.
Step 24.

$$\min_{W_{23}^{'} - W_{23} = 1} h_d(E_{24}^{'}, E_{24}) = \min_{W_{23}^{'} - W_{23} = 1} h_d(C_1 + 1, C_1) = 1,$$

where $C_1 = \Sigma_1(E_{23}) + Ch(E_{23}, F_{23}, G_{23}) + H_{23} + D_{23} + K_{23} + W_{23}$.

$$\min_{W_{23}^{'} - W_{23} = 1} h_d(A_{24}^{'}, A_{24}) = \min_{W_{23}^{'} - W_{23} = 1} h_d(C_2 + 1, C_2) = 1,$$

where $C_2 = \Sigma_0(A_{23}) + Maj(A_{23}, B_{23}, C_{23}) + \Sigma_1(E_{23}) + Ch(E_{23}, F_{23}, G_{23}) + H_{23} + K_{23} + W_{23}$.
We have the minimal Hamming distances when $C_1^{32} = C_2^{32} = 0$, which means with probability 2^{-2}.

Step 25.

$$\min_{W_{24}^{'} - W_{24} = -1 + \Delta\sigma_0(1)} h_d(E_{25}^{'}, E_{25}) =$$

$= \min h_d(\Sigma_1(E'_{24}) + Ch(E'_{24}, F_{24}, G_{24}) - 1 + \sigma_0(m_9+1) + C_1, \Sigma_1(E_{24}) + Ch(E_{24}, F_{24}, G_{24}) + \sigma_0(m_9) + C_1),$

where $C_1 = H_{24} + D_{24} + K_{24} + \sigma_1(W_{22}) + m_8$. If $F^{32}_{24} = 1$ and $G^{32}_{24} = 0$ (probability 2^{-2}) then, considering that $E'^{32}_{24} = 1, E^{32}_{24} = 0$, we have $Ch(E'_{24}, F_{24}, G_{24}) - 1 = Ch(E_{24}, F_{24}, G_{24})$, and we can rewrite the last expression as:

$$\min h_d(\Sigma_1(E'_{24}) + \sigma_0(m_9 + 1) + C_2, \Sigma_1(E_{24}) + \sigma_0(m_9) + C_2),$$

where $C_2 = C_1 + Ch(E_{24}, F_{24}, G_{24})$.
If no carry occurs due to the differences, then the above minimum is:

$$\min h_d(\Sigma_1(E'_{24}) + \sigma_0(m_9 + 1) + C_2, \Sigma_1(E_{24}) + \sigma_0(m_9) + C_2) = 5.$$

For $\Sigma_1(E'_{24})$ (difference in three bits) there are no carries with probability 2^{-3}. For $\sigma_0(m_9 + 1)$ (two differences if $m^{32}_9 = 0$) with probability 2^{-3}. Therefore the minimum is 5 with probability 2^{-8}.
Using the same methods we can get:

$$\min_{W'_{24} - W_{24} = -1 + \Delta\sigma_0(1)} h_d(A'_{25}, A_{25}) = 8,$$

with probability 2^{-11}. Notice that if minimum holds for A_{25} then it holds for E_{25}. So, for the whole hash value, we have:

$$\begin{aligned}
&h_d((A'_{25}, B'_{25}, C'_{25}, D'_{25}, E'_{25}, F'_{25}, G'_{25}, H'_{25})), (A_{25}, B_{25}, C_{25}, D_{25}, E_{25}, F_{25}, G_{25}, H_{25})) = \\
&= h_d((A'_{25}, A'_{24}, C_{25}, D_{25}, E'_{25}, E'_{24}, G_{25}, H_{25}), (A_{25}, A_{24}, C_{25}, D_{25}, E_{25}, E_{24}, G_{25}, H_{25})) = \\
&= h_d(A'_{25}, A_{25}) + h_d(E'_{25}, E_{25}) + h_d(A'_{24}, A_{24}) + h_d(E'_{24}, E_{24}) = \\
&= 8 + 5 + 1 + 1 = 15
\end{aligned}$$

Therefore we get a 25-step semi-free start near collision with the Hamming weight of 15 bits and complexity of 2^{34} SHA-2 compression function calls. Notice that we have not investigated all the possible outcomes of the carry effects. Therefore, it is possible that the real complexity is lower.

Conditions for Collisions and Collision Examples

Table 6.3: The differences propagation for 20, 21, and 23-step collisions for SHA-256. Notice that for each collision initial difference is introduced in different steps (steps 5,6,9 respectively).

20 step	21 step	23 step	ΔA	ΔB	ΔC	ΔD	ΔE	ΔF	ΔG	ΔH	ΔW
5	6	9	0	0	0	0	0	0	0	0	1
6	7	10	1	0	0	0	1	0	0	0	δ_1
7	8	11	0	1	0	0	-1	1	0	0	δ_2
8	9	12	0	0	1	0	0	-1	1	0	δ_3
9	10	13	0	0	0	1	0	0	-1	1	0
10	11	14	0	0	0	0	1	0	0	-1	0
11	12	15	0	0	0	0	0	1	0	0	0
12	13	16	0	0	0	0	0	0	1	0	δ_5
13	14	17	0	0	0	0	0	0	0	1	-1
14	15	18	0	0	0	0	0	0	0	0	0

Table 6.4: The values of the word differences in 20, 21, and 23-step collisions for SHA-256. Notice that 23-step semi-free start collision has a word difference in δ_5. That is why its collision path is slightly different than the one used for 20 and 21-step collisions.

20-step	$\delta_1 = -1 - \Delta Ch^6(1,0,0) - \Delta\Sigma_1(E_6)$ $\delta_2 = -\Delta\Sigma_1(E_7) - \Delta Ch^7(-1,1,0)$ $\delta_3 = -\Delta Ch^8(0,-1,1)$ $\delta_5 = 0$
21-step	$\delta_1 = -1 - \Delta Ch^7(1,0,0) - \Delta\Sigma_1(E_7)$ $\delta_2 = -\Delta\Sigma_1(E_8) - \Delta Ch^8(-1,1,0)$ $\delta_3 = -\Delta Ch^9(0,-1,1)$ $\delta_5 = 0$
23-step	$\delta_1 = -1$ $\delta_2 = -\Delta\Sigma_1(E_{11}) - \Delta Ch^{11}(-1,1,0)$ $\delta_3 = -\Delta Ch^{12}(0,-1,1)$ $\delta_5 = 1$

Table 6.5: The additional conditions that have to hold in order to get a 20, 21, and 23-step collisions for SHA-256.

20-step	$A_4 = A_5 = A_7 = A_8$ $A_6 = -1, A_6^{'} = 0$	$E_9 = E_8, E_{10} = -1, E_{10}^{'} = 0$ $E_{11} = 0, E_{12} = -1$	$\Delta Ch^9(0,0,-1) = -1$
21-step	$A_5 = A_6 = A_8 = A_9$ $A_7 = -1, A_7^{'} = 0$	$E_{10} = E_9, E_{11} = -1, E_{11}^{'} = 0$ $E_{12} = 0, E_{13} = -1$	$\Delta Ch^{10}(0,0,-1) = -1$ $\Delta\sigma_1(-1) + \delta_3 = 0$
23-step	$A_8 = A_6 = A_9 = A_{10}$ $A_{10} = -1, A_{10}^{'} = 0$	$E_{13} = E_{12}, E_{14} = -1, E_{14}^{'} = 0$ $E_{15} = 0, E_{16} =$0xfffffffe $E_9 = E_8 + 1, E_{10} = -1, E_{10}^{'} = 0$	$\Delta Ch^{13}(0,0,-1) = -1$ $\Delta\sigma_1(-1) + \delta_3 = 0$ $\Delta\sigma_1(1) + \delta_2 = 0$

Table 6.6: A 21-step collision for SHA-256

M_0	0004024f	00000000	00000000	00000000
	00000000	2c51fd8d	b83daf3c	bc852709
	ae18a3e7	1d11dbc7	21d06175	ab551b5f
	a48e9a8b	00000000	19000000	00000000
$M_0^{'}$	0004024f	00000000	00000000	00000000
	00000000	2c51fd8d	b83daf3d	7c652ab7
	b238a344	1d11dac8	21d06175	ab551b5f
	a48e9a8b	00000000	18ffffff	00000000
H	73f5fcd2	682f578e	8d9c3d05	f93ad865
	662b0636	a5a5d4c2	32091775	04ac6dae

Table 6.7: A 23-step semi-free start collision for SHA-256

H_0	cb518aaa	55d8f4ad	231e476a	89ac8889
	f29c30cc	2e1f63c5	cf4f2366	75367200
M_0	b5c16a2d	6da1708b	00000000	00000000
	00000000	00000000	00000000	00000000
	00000000	a9d5faeb	54eb8149	085be1ce
	b9e61e60	9380ae01	efa5a517	cdc5da00
$M_0^{'}$	b5c16a2d	6da1708b	00000000	00000000
	00000000	00000000	00000000	00000000
	00000000	a9d5faec	54eb8148	085c0205
	b9e61d61	9380ae01	efa5a517	cdc5da00
H	6682cc14	9c825293	bc17ea6d	d89770cf
	a69ac7ed	cfa5ee3e	e35c0091	7249d71e

Table 6.8: A 25-step semi-free start near collision with Hamming distance of 17 bits for SHA-256.

H_0	8e204f9e	bca27aea	42da63d7	00f2f219
	fd1db715	6389ae13	c6f57538	de4e655c
M_0	c63714eb	13d5fa9c	00000000	00000000
	00000000	00000000	00000000	00000000
	00000000	d51b4dba	aeb6f738	61dce9b7
	0ab5c01a	83406f01	df65666b	cdc5da00
$M_0^{'}$	c63714eb	13d5fa9c	00000000	00000000
	00000000	00000000	00000000	00000000
	00000000	d51b4dbb	aeb6f737	71dd499a
	0ab5bf1b	83406f01	df65666b	cdc5da00
H	2e2fcb73	8192d3a4	f85b5a7d	801c4583
	9307e51c	cf57fb61	11c48b0d	7131ccd2
$H^{'}$	6c478ef3	8192d3a5	f85b5a7d	801c4583
	9127a49c	cf57fb62	11c48b0d	7131ccd2

Chapter 7

Pseudo-Collisions for LAKE

The most widely used hash function construction – the Merkle-Damgård construction, has some undesirable security properties such as the long message second-preimage attack found by Dean [47] and Kelsey-Schneier [73], and the herding attack by Kelsey and Kohno [71]. Therefore, alternative methods for designing hash functions from compression functions are greatly appreciated. One of these methods is the HAIFA construction [19]. The hash function LAKE [10] was the first hash based on the HAIFA design. It is a software-orientated hash that supports 256-bit and 512-bit outputs. It is supposed to be flexible, i.e. by increasing the number of internal rounds of the compression function the security should also increase. Important point is that LAKE is faster than SHA-2, which was considered to be the unofficial new standard after the attacks of SHA-1 and MD5 [140, 141].

The first analysis of LAKE, collisions for 4 out of 8 rounds with complexity of 2^{109}, was published by Mendel and Schläffer [99]. The main idea used in the attack is the non-injectivity of one of the internal updating functions. This property allows to introduce difference in message words which is canceled immediately when the difference goes through the non-injective function. The authors show that in slightly modified version of LAKE, where all constants are the same (not necessary equal to zero), or the constants have some different but specially chosen values, it is possible to find collisions for all 8 rounds. However, the unmodified version of LAKE, cannot be attacked on more than 5 of the 8 rounds, when this technique is applied.

Our attacks focus on finding pseudo-collisions in the compression function of LAKE. In both of our attacks, we attack the full, unmodified compression function. In the first attack we introduce differences in the chaining value and the block index, but not in the message words. In the second attack we introduce differences only in the chaining values. These differences are canceled right after the first round of the internal processmessage procedure, so the attack is applicable to any number of rounds and its complexity stays the same. To cancel the differences, a few equations with very similar structure are obtained. We were able to find good algorithms for solving these equations and additionally decrease the total complexities of the attacks. For finding a pseudo collision for LAKE-256, with differences

in the initial value and the block index, the complexity of our algorithm is about 2^{33} compression function calls. Our second algorithm finds pseudo collisions in LAKE-224 with difference only in the initial value, with complexity of 2^{99} calls and can be trivially extended to a pseudo collision attack on the whole hash design.

A pseudo collision attack on the compression function of LAKE was as well presented in [27]. The attack utilizes differences in the chaining values and salt and yields collisions with complexity 2^{42} calls.

7.1 Description of LAKE

The hash function family LAKE supports two basic variants: LAKE-256 and LAKE-512. Other digest variants are obtained by truncation of the basic variants. Further, we will describe only LAKE-256; the definition of LAKE-512 can be found in [10]. The variants differ only in the word size, constants used and rotation parameters. LAKE is based on HAIFA design. It takes a message and a salt as an input, and outputs 256-bit hash. First, the standard padding is applied to the message, then the message is divided into 512-bit blocks, and each block is processed, iteratively, by the compression function of LAKE. This function takes the previous chaining value h_{l-1}, the current message block M_l, the salt S, and the current block index t_l and outputs a new chaining value h_l, i.e. $h_l = compress(h_{l-1}, M_l, S, t_l)$. The initial chaining value is fixed. The output of the compression function processing the last block of the message is the hash of the whole message.

Basically, all the transformations in the compression function are done by two non-linear functions:

$$f(a,b,c,d) = [a + (b \vee C_0)] + ([c + (a \wedge C_1)] \ggg 7) + ([b + (c \oplus d) \ggg 13)$$
$$g(a,b,c,d) = [(a+b) \ggg 1] \oplus (c+d)$$

All the operations are on 32-bit words. With $+$ is denoted addition mod 2^{32}, $\oplus$ is bitwise XOR, $\ggg$ is cyclic rotation to the right, and C_i are constants.

The compression function itself, can be divided into three parts: initialization (saltstate), internal round function (processmessage), and finalization (feedforward). Further, you can find the pseudo-code of all three parts.

Initialization - Saltstate

input $h = H_0 \| H_1 \| \ldots H_7, \quad S = S_0 \| \ldots \| S_3, \quad t = t_0 \| t_1$
1. for $i = 0, \ldots, 7$ do
   $L_i \leftarrow H_i$
2. $L_8 \leftarrow g(H_0, S_0 \oplus t_0, C_8, 0)$
3. $L_9 \leftarrow g(H_1, S_1 \oplus t_1, C_9, 0)$
4. for $i = 10, \ldots, 15$ do
   $L_i \leftarrow g(H_i, S_i, C_i, 0)$

output $L = L_0 \| L_1 \| \ldots L_{15}$

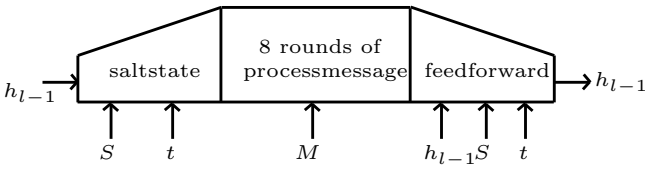

Figure 7.1: The compression function for LAKE. Starting from left to the right, are the three procedures: initialization, round function and finalization.

Round function - Processmessage

input $L = L_0 \parallel L_1 \parallel \ldots L_{15}, \quad M = M_0 \parallel M_1 \parallel \ldots \parallel M_{15}, \quad \sigma$
1. $F \leftarrow L$
2. for $i = 0, \ldots, 15$ do
   $L_i \leftarrow f(L_{i-1}, L_i, M_{\sigma(i)}, C_i)$
3. for $i = 0, \ldots, 15$ do
   $L_i \leftarrow g(L_{i-1}, L_i, F_i, L_{i+1})$
output $L = L_0 \parallel L_1 \parallel \ldots L_{15}$

Finalization - Feedforward

input $L = L_0 \parallel \ldots L_{15} \quad h = H_0 \parallel H_1 \parallel \ldots H_7, \quad S = S_0 \parallel \ldots \parallel S_3, \quad t = t_0 \parallel t_1$
1. $H_0 \leftarrow f(L_0, L_8, S_0 \oplus t_0, H_0)$
3. $H_1 \leftarrow f(L_1, L_9, S_1 \oplus t_1, H_1)$
4. for $i = 2, \ldots, 7$ do
   $H_i \leftarrow f(L_i, L_{i+8}, S_i, H_i)$
output $h = H_0 \parallel H_1 \parallel \ldots H_7$

In the processmessage procedure, $\sigma(x)$ is a permutation. The exact definition of this permutation, as well as the values of the constants can be found in [10]. The additions in the indexes for L_i, H_i, and S_i are done mod 16, 8, and 4 respectively.

In short, first the chaining value is expanded, by the saltstate procedure, from 8 words to 16 words. This becomes the internal state. After that, the round function is applied to the internal state 8 times (in the above algorithm of the round function, only one round is defined). At the end, the feedforward procedure is applied, and the internal state is reduced into 8 words chaining value.

7.2 Analysis of LAKE

The input of the compression function of LAKE is the previous chaining value h_{l-1}, the message M_l, the block index t_i, and the salt S. Our first pseudo collision attack,

further referred as **PSA1**, finds two different chaining values h^1 and h^2 and two different block indexes t^1 and t^2 such that:

$$compress(h^1, M, S, t^1) = compress(h^2, M, S, t^2), \tag{7.1}$$

for some message block M and salt S. Notice that there are no differences in the message words. The salt can be chosen freely.

Our second attacks, further referred as **PSA2**, finds two different chaining values h_1 and h_2 such that:

$$trunc_r(compress(h^1, M, S, t)) = trunc_r(compress(h^2, M, S, t)), \tag{7.2}$$

where $trunc_r(x)$ is function that returns the last 224 bits of x (truncation to the right) and M, S, t are some values. Hence, the second attack finds a near pseudo collisions for the compression function with differences only in the first word of the output. In a truncated version, like LAKE-224, this would mean a collision[1].

The rest of the analysis has the following outline. First, we will point out some properties of the non-linear functions used in LAKE. Using these properties we will launch the first pseudo collision attack **PSA1** for the middle building block of the compression function, the processmessage function, and then we will extend the attack to the saltstate and feedforward functions as well. Further, we will find the solutions for the equations we get in the meantime, and we will estimate the total complexity of the attack. In a similar way as the first attack **PSA1**, we will present the second pseudo collision attack **PSA2**.

7.2.1 Simple Observations

From the design of LAKE, few simple observations follow. These observations are directly used in the attacks of the compression function.

Observation 1 *Function $f(x, y, z, t)$ is non-injective by the first three arguments x, y, z.*

For example, for x there exist different values x_1,x_2 such that $f(x_1, y, z, t) = f(x_2, y, z, t)$ for some y, z, t. The same property holds for y and z.

This observation was mentioned by Lucks at FSE'08. Mendel and Schläffer independently found and used this property to successfully attack four out of eight rounds of LAKE-256. Non-injectivity of the function f can be used to cancel a difference in one of the first three arguments of f, when the rest of the arguments are properly fixed.

Observation 2 *There is some non-symmetry when updating the state variables.*

It is important to notice when the internal variables are updated one by one, by the functions f and g, there are no temporary variables that store their previous values. This is supposedly done in order to increase the diffusion. Yet, it leads to

[1] In the submission paper the truncation is to the left.

some rather non-symmetrical situations. For example, when L_i is being updated by the function f, its new value depends, particularly, of the new value of the previous variable L_{i-1}. In other words, first L_{i-1} is computed, and then this value is used to calculate the new value of L_i. But, when L_0 is updated by the function f, it depends on the "old" value of L_{15}, because L_{15} is updated last. Same holds when the function g is used as an updating function. When L_i is being updated by the function g, its value depends, particularly, on the new value of previous variable L_{i-1} and the old value of the next variable L_{i+1}. But, when L_{15} is updated, it takes the new value of the next variable, which is L_0. This implies that the first and the last variables are updated according to slightly different rules.

7.2.2 PSA1 on Processmessage Procedure

First, let us try to attack only the middle procedure of the compression function, i.e. processmessage function. It consists of 8 rounds (10 rounds for LAKE-512). In every round, first all of the 16 internal variables are updated by the function f, and then all of them are updated by the function g.
Let T_i be the updated, by the function f, value of the variable L_i.
Let N_i be the updated, by the function g, value of the variable T_i (or same as updated value of L_i by both f and g).

Our attack strategy is the following (Fig. 7.3):

1. Let only L_0 contain some specially chosen non-zero difference.
2. After the application of the function f we require zero differences in all the variables except T_0.
3. After the application of the function g we require zero differences in all the variables.

Let us show that this scenario is possible. First let us prove that step 2 is achievable. Considering that $T_i = f(T_{i-1}, L_i, M_i, C_i)$ we get that in T_i a difference can be introduced only through T_{i-1} and L_i (message words do not have differences (7.1), C_i are simply constants).
For ΔT_0 we require non-zero difference:

$$\Delta T_0 = f(L_{15}, L_0^2, M_0, C_0) - f(L_{15}, L_0^1, M_0, C_0) \neq 0 \tag{7.3}$$

Although in observation 1 we mentioned that the function f is non-injective by the second argument, it is still easy to find different values for L_0 that lead to different values for f. The first argument of f is not T_{15} but L_{15} (as mentioned in the observation 2).
For ΔT_1 we require zero difference:

$$\Delta T_1 = f(T_0^2, L_1, M_1, C_1) - f(T_0^1, L_1, M_1, C_1) = 0 \tag{7.4}$$

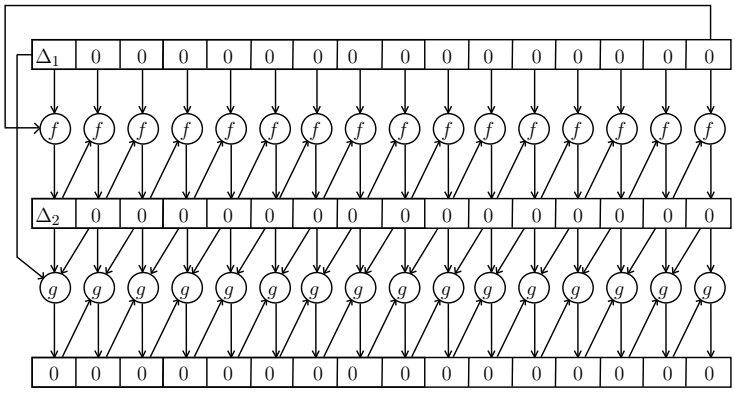

Figure 7.2: Differences in the variables for the PSA1 on processmessage. The zero differences coming from M_i and C_i, as well as zero differences coming from $F_i (i > 0)$ are not shown.

By observation 1 it follows that it is possible to get zero for ΔT_1.
For all other variables we require zero difference:

$$\Delta T_i = f(T_{i-1}, L_i, M_i, C_i) - f(T_{i-1}, L_i, M_i, C_i) = 0$$

Note that all the variables are the same, there are no differences in T_i, thus these equations trivially hold.
Now, let us take a look at step 3. Again, considering that $N_i = g(N_{i-1}, T_i, L_i, T_{i+1})$, we get that in N_i a difference can be introduced by any of N_{i-1}, T_i, L_i and T_{i+1}.
For ΔN_0 we require zero difference, so we get:

$$\Delta N_0 = g(T_{15}, T_0^2, L_0^2, T_1) - g(T_{15}, T_0^1, L_0^1, T_1) = 0 \tag{7.5}$$

Note that there are differences in two variables, T_0 and L_0, and even though g is an invertible function, it is still possible to solve this equation.
For the indexes $i = 1..14$ we get:

$$\Delta N_i = g(N_{i-1}, T_i, L_i, T_{i+1}) - g(N_{i-1}, T_i, L_i, T_{i+1}) = 0 \tag{7.6}$$

All the equations hold because there are no difference in any of the arguments.
For N_{15} we get:

$$\Delta N_{15} = g(N_{14}, T_{15}, L_{15}, N_0) - g(N_{14}, T_{15}, L_{15}, N_0) = 0$$

Notice that the last argument is not T_0 but rather N_0. This is where the observation 2 is applied. Non-symmetry allows us to get that this equation also holds if the previous equations hold.

So, after only one round we can obtain an internal state with all-zero differences in the variables. But then, all the following rounds can not introduce any difference because there are zero differences in the internal state variables and zero

differences in the message words (7.1). So, if we are able to solve the equations that we got then this means that *the attack is applicable to any number of rounds, i.e. increasing the number of rounds does not improve the security of* processmessage *function of LAKE.*
Now let us take a closer look at the equations that we got.
Equation (7.3) can be rewritten as:

$$\begin{aligned}\Delta T_0 &= f(L_{15}, L_0^2, M_0, C_0) - f(L_{15}, L_0^1, M_0, C_0) = \\ &= (L_0^2 \vee C_0) - (L_0^1 \vee C_0) + [L_0^2 + (M_0 \oplus C_0)] \gg 13 - [L_0^1 + (M_0 \oplus C_0)] \gg 13\end{aligned}$$

Here and further we will use that $(A+B) \gg r = (A \gg r) + (B \gg r)$ with probability around $\frac{1}{4}$ (same holds when shift to the left is used). Therefore the above equation can be rewritten as:

$$\Delta T_0 = (L_0^2 \vee C_0) - (L_0^1 \vee C_0) + L_0^2 \gg 13 - L_0^1 \gg 13 \tag{7.7}$$

Equation (7.4) can be written as:

$$\begin{aligned}\Delta T_1 =& f(T_0^2, L_1, M_1, C_1) - f(T_0^1, L_1, M_1, C_1) = \\ =& T_0^2 - T_0^1 + [M_1 + (T_0^2 \wedge C_1)] \gg 7 - [M_1 + (T_0^1 \wedge C_1)] \gg 7 = \\ =& T_0^2 - T_0^1 + (T_0^2 \wedge C_1) \gg 7 - (T_0^1 \wedge C_1) \gg 7 = 0\end{aligned}$$

Equation (7.5) can be written as:

$$\begin{aligned}\Delta N_0 =& g(T_{15}, T_0^2, L_0^2, T_1) - g(T_{15}, T_0^1, L_0^1, T_1) = \\ =& [(T_{15} + T_0^2) \gg 1g] \oplus (L_0^2 + T_1) - [(T_{15} + T_0^1) \ggg 1] \oplus (L_0^1 + T_1) = 0\end{aligned}$$

The details of solving these equations are presented in Section 7.2.4.

7.2.3 PSA1 for the Full Compression Function

The designers of LAKE specifically stated that any attack on LAKE should be on the whole design and not on the separate parts of the compression function. This way the good properties of the HAIFA design are brought to the forefront. Therefore let us try to attack the whole design. So far, if we assume that we can solve the previous equations, we can get a pseudo-collision attack on processmessage function. Let us try to extend the attack to the whole compression function. First, let us deal with the initialization (function saltstate).

From the initialization of LAKE it can be seen that the variables H_0 through H_7 are straightforward copied into L_0 through L_7. The variable L_8 depends on H_0 and t_0. Similarly, L_9 depends on H_1 and t_1. The rest of the variables do not depend on the t_0 or t_1. Again, we can see some additional non-symmetry apart from the already mentioned in the observation 2. Since we need a difference in L_0 (for the previous attack on processmessage function), we will introduce difference in H_0. Further we can follow our previous attack on the processmessage block and get a collisions after the processmessage function. The only difficulty is how to deal

with L_8 since it does depend on H_0 which now has a non-zero difference. To our help comes the block index t_0. By introducing a difference in t_0 we can cancel the difference from H_0 in L_8. So we get the following equation:

$$\begin{aligned}\Delta L_8 &= g(H_0^2, S_0 \oplus t_0^2, C_0, 0) - g(H_0^1, S_0 \oplus t_0^1, C_0, 0) = \\ &= ((H_0^2 + (S_0 \oplus t_0^2)) \ggg 1 \oplus C_0) - ((H_0^1 + (S_0 \oplus t_0^1)) \ggg 1 \oplus C_0) = 0\end{aligned}$$

Let $\tilde{t_0^2} = t_0^2 \oplus S_0$ and $\tilde{t_0^1} = t_0^1 \oplus S_0$. Then, the above equation gets the following form:

$$\Delta L_8 = H_0^2 - H_0^1 + \tilde{t_0^2} - \tilde{t_0^1} = 0$$

Now, let us deal with the last building block of the compression function, the finalization (feedforward). We keep in mind that we have differences only in H_0 and t_0. If we take a glance at the feedforward procedure, we can see that H_0 and t_0 can be found in the same equation, and only there, which defines the new value for H_0. Since, we require zero difference in all of the output variables, we get the following equation:

$$\begin{aligned}\Delta H_0 &= f(L_0, L_8, H_0^2, S_0 \oplus t_0^2) - f(L_0, L_8, H_0^1, S_0 \oplus t_0^1) = \\ &= \tilde{t_0^2} \gg 7 - \tilde{t_0^1} \gg 7 + (\tilde{t_0^2} \oplus H_0^2) \gg 13 - (\tilde{t_0^1} \oplus H_0^1) \gg 13 = 0\end{aligned}$$

This concludes our attack. We have shown that if we introduce a difference only in the chaining value H_0 and the block index t_0, it is possible to reduce the problem of finding pseudo collisions for the compression function to the problem of solving a system of equations.

7.2.4 Solution of the System of Equations

To find a pseudo collision for the full compression function of LAKE, we have to solve the equations that were mentioned in the previous sections. As a result, we get the following system:

$$T_0^2 - T_0^1 + (T_0^2 \wedge C_1) \gg 7 - (T_0^1 \wedge C_1) \gg 7 = 0 \tag{7.8}$$

$$T_0^2 - T_0^1 = (H_0^2 \vee C_0) - (H_0^1 \vee C_0) + H_0^2 \gg 13 - H_0^1 \gg 13 \tag{7.9}$$

$$[(T_{15} + T_0^2) \gg 1] \oplus (H_0^2 + T_1) - [(T_{15} + T_0^1) \gg 1] \oplus (H_0^1 + T_1) = 0 \tag{7.10}$$

$$H_0^2 - H_0^1 + \tilde{t_0^2} - \tilde{t_0^1} = 0 \tag{7.11}$$

$$\tilde{t_0^2} \gg 7 - \tilde{t_0^1} \gg 7 + (\tilde{t_0^2} \oplus H_0^2) \gg 13 - (\tilde{t_0^1} \oplus H_0^1) \gg 13 = 0 \tag{7.12}$$

Let us analyze (7.8). By fixing $T_0^2 - T_0^1 = R$, this equation can be rewritten as:

$$R + [(T_0^1 + R) \wedge C_1] \gg 7 - (T_0^1 \wedge C_1) \gg 7 = 0 \tag{7.13}$$

If we rotate everything to the left by 7 bits we get:

$$[(T_0^1 + R) \wedge C_1] - (T_0^1 \wedge C_1) = (-R) \ll 7 \tag{7.14}$$

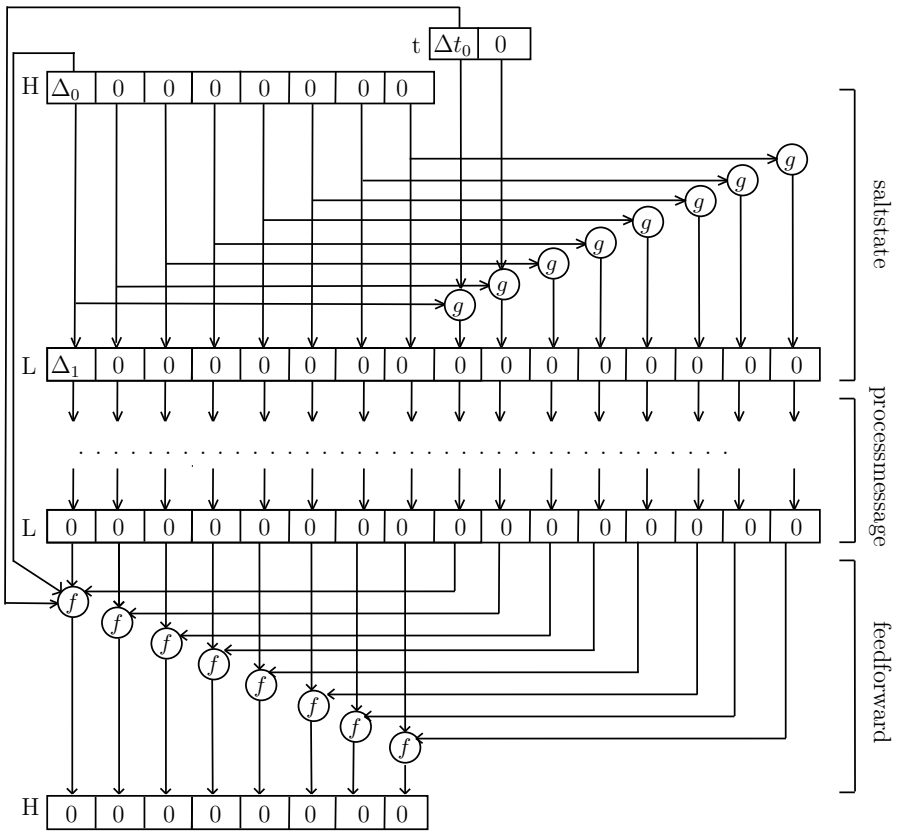

Figure 7.3: Differences in the variables for **PSA1** on the full compression function. The zero differences coming from M_i and C_i, as well as zero differences coming from $F_i (i > 0)$ are not pictured.

As a result, we get an equation of the following form:

$$(X + A) \wedge C = X \wedge C + B \tag{7.15}$$

where $X = T_0^1, A = R, B = (-R) \ll 7, C = C_1$.

Now, let us analyze (7.9). Again, let us fix $T_0^2 - T_0^1 = R$ and $H_0^2 - H_0^1 = D$. Then (7.9) gets the following form:

$$R = ((H_0^1 + D) \vee C_0) - (H_0^1 \vee C_0) + D \gg 13 \tag{7.16}$$

This equation can be rewritten as:

$$(X + A) \vee C = X \vee C + B \tag{7.17}$$

where $X = H_0^1, A = D, B = R - (D \gg 13), C = C_0$.

In (7.10), if we regroup the components, we get:

$$[(T_{15} + T_0^2) \oplus (T_{15} + T_0^1)] \ggg 1 = (H_0^2 + T_1) \oplus (H_0^1 + T_1)$$

Then, the above equation is of the following form:

$$((X + A) \oplus X) \ggg 1 = (Y + B) \oplus Y, \tag{7.18}$$

where $X = T_{15} + T_0^1, A = T_0^2 - T_0^1, Y = T_1 + H_0^1, B = H_0^2 - H_0^1$.

Now, let us analyze (7.11) and (7.12). Let us fix $H_0^2 - H_0^1 = D$. Note that then from (7.11) we have $\tilde{t_0^2} - \tilde{t_0^1} = -D$. If we rotate everything by 13 bits to the left in (7.12) we get:

$$(-D) \ll 6 + (\tilde{t_0^2} \oplus H_0^2) - (\tilde{t_0^1} \oplus H_0^1) = 0 \tag{7.19}$$

$$\tilde{t_0^1} = [(\tilde{t_0^2} \oplus H_0^2) - D \ll 6] \oplus H_0^1 \tag{7.20}$$

If we put this expression for $\tilde{t_0^1}$ in (7.11) we get:

$$D + \tilde{t_0^2} - [(\tilde{t_0^2} \oplus H_0^2) - D \ll 6] \oplus H_0^1 = 0 \tag{7.21}$$

$$\tilde{t_0^2} = [(\tilde{t_0^2} \oplus H_0^2) - D \ll 6] \oplus H_0^1 - D \tag{7.22}$$

If we XOR the value of H_0^2 to the both sides:

$$\tilde{t_0^2} \oplus H_0^2 = ([(\tilde{t_0^2} \oplus H_0^2) - D \ll 6] \oplus H_0^1 - D) \oplus H_0^2 \tag{7.23}$$

Let us denote $\tilde{t_0^2} \oplus H_0^2 = X$. Then we get:

$$X = [(X - D \ll 6) \oplus H_0^1 - D] \oplus H_0^2 \tag{7.24}$$

$$X \oplus H_0^2 = (X - D \ll 6) \oplus H_0^1 - D \tag{7.25}$$

Finally, we get an equation of the following form:

$$(X \oplus K_1) + A = (X + B) \oplus K_2 \tag{7.26}$$

where $K_1 = H_0^2, A = R, B = -R \ll 6, K_2 = H_0^1$.

There exist efficient algorithms (we call them **Al1**,**Al2**,**Al3**,**Al4**) for finding solutions for equations of type (7.15),(7.17),(7.18),(7.26). In the following lemmas we prove there existence.

Lemma 1 *There exist an algorithm (**Al1**) for finding all the solutions for the equation of type (7.15). The complexity of **Al1** depends only on the constant C.*

Lemma 2 *There exist an algorithm (**Al2**) for finding all the solutions for the equation of type (7.17). The complexity of **Al2** depends only on the constant C.*

Proof. The proofs for the two facts are very similar with some minor changes, so we will prove only Lemma 1.
Let $X = x_{31}\dots x_1x_0, A = a_{31}\dots a_1a_0, B = b_{31}\dots b_1b_0, C = c_{31}\dots c_1c_0$. Then for each i we have:

$$(x_i \wedge c_i) \oplus a_i \oplus t_i = (x_i \oplus b_i \oplus r_i) \wedge c_i, \tag{7.27}$$

where $t_i = m(x_{i-1} \wedge c_{i-1}, a_{i-1}, t_{i-1})$ is the carry at the $(i-1)$th position of $(X \wedge C + A)$, $r_i = m(x_{i-1}, b_{i-1}, r_{i-1})$ is the carry at the $(i-1)$th position of $X+B$, and $m(x,y,z) = xy \oplus xz \oplus yz$.
The equation (7.27), when $c_i = 0$, gets the following form:

$$a_i \oplus t_i = 0. \tag{7.28}$$

When $c_i = 1$, we get:

$$a_i \oplus t_i = b_i \oplus r_i \tag{7.29}$$

Let us assume that we found the values for t_i and r_i for some i. We find the smallest $j > 0$ such that $c_{i+j} = 0$. Then from (7.28) and the definition of t_i we get:

$$\begin{aligned} a_{i+j} =& t_{i+j} = m(x_{i+j-1}, a_{i+j-1}, t_{i+j-1}) = \\ =& m(x_{i+j-1}, a_{i+j-1}, m(x_{i+j-2}, a_{i+j-2}, t_{i+j-2})) = \dots \\ =& m(x_{i+j-1}, a_{i+j-1}, m(x_{i+j-2}, a_{i+j-2}, m(\dots, m(x_i, a_i, t_i))\dots)) \end{aligned}$$

In the above equation, only $x_i, x_{i+1}, \dots x_{i+j-1}$ are unknown. So we can try all the possibilities, which are 2^j, and find all the solutions. Let us denote by $\tilde{X}$ the set of all solutions.
Now, let us find the smallest $l > 0$ such that $c_{i+j+l} = 1$. Notice that we can easily find t_{i+j+1} if considering $c_{i+j+l_0} = 0$ for $l_0 \in (0, l)$ and using (7.28):

$$\begin{aligned} t_{i+j+1} =& m(0, a_{i+j}, t_{i+j}) = m(0, a_{i+j}, a_{i+j}) = a_{i+j} \\ t_{i+j+2} =& m(0, a_{i+j+1}, t_{i+j+1}) = m(0, t_{i+j+1}, t_{i+j+1}) = m(0, a_{i+j}, a_{i+j}) = a_{i+j} \\ & \dots \\ t_{i+j+l} =& m(0, a_{i+j+l-1}, t_{i+j+l-1}) = a_{i+j} \end{aligned}$$

From (7.29) and definition of r_i we get:

$$\begin{aligned} a_{i+j+l} \oplus t_{i+j+l} \oplus b_{i+j+l} = r_{i+j+l} =& m(x_{i+j+l-1}, b_{i+j+l-1}, r_{i+j+l-1}) = \\ =& m(x_{i+j+l-1}, b_{i+j+l-1}, m(x_{i+j+l-2}, b_{i+j+l-2}, r_{i+j+l-2})) = \dots \\ =& m(x_{i+j+l-1}, b_{i+j+l-1}, m(\dots, m(x_i, b_i, r_i)\dots)) \end{aligned}$$

In the above equation, only $x_i, x_{i+1}, \ldots, x_{i+j+l-1}$ are unknown. Hence we check all the possibilities by taking $(x_i, x_{i+1}, \ldots, x_{i+j-1})$ from the set $\tilde{X}$ and the rest of the variables take all the possible values. If the equation has a solution, then this means we have fixed another t_{i+j+l}, r_{i+j+l}, and we can continue searching using the same algorithm.
The complexity of the algorithms is 2^q, where q is size of the longest consecutive sequence of ones followed by consecutive zero sequence (in the case above $q = j+l$) in the constant C. Taking into consideration the value of the constant C_1 used in the compression function of LAKE-256, we get that complexity of our algorithm for this special case is 2^8. Yet, the average complexity can be decreased additionally if first the necessary conditions are checked. For example, if we have two consecutive zeros in the constant C_1 at positions i and $i+1$ then it has to hold $a_{i+1} = a_i$. If we check for all zeros, then only with probability of 2^{-10} a constant A can pass this sieve. Therefore, the math expectancy of the complexity for a random A is less than 2^1.
Note that when $\vee$ function is used instead of $\wedge$, than 0 and 1 change place. Therefore, our algorithm has a complexity of 2^6 when C_0 is used as a constant. Yet, same as for $\wedge$, early break-up strategies significantly decrease these complexities for the case when solution does not exist. Again, the average complexity is less than 2^1. □

Lemma 3 *There exist an algorithm (**Al3**) for finding a solution for the following equation:*

$$((X+A) \oplus X) \gg 1 = (Y+B) \oplus Y \tag{7.30}$$

Proof. Instead of finding a solution with respect to X and Y we split (7.30) into a system:

$$(X+A) \oplus X = -1 \tag{7.31}$$

$$(Y+B) \oplus Y = -1 \tag{7.32}$$

We can do this because the value of -1 is invariant of any rotation. We may loose some solutions, but further we will prove that if such a solution exist then our algorithm will find it with probability 2^{-2}.
We will analyze only (7.31); obviously the second equation can be solved analogously.
Let $X = x_{31} \ldots x_1 x_0, A = a_{31} \ldots a_1 a_0$. Then for ith bit we get:

$$(x_i \oplus a_i \oplus c_i) \oplus x_i = 1,$$

where c_i is the carry at $(i-1)$ position of $X+A$, i.e. $c_i = m(x_{i-1}, a_{i-1}, c_{i-1})$. Obviously, the above equation can be rewritten as:

$$a_i = c_i \oplus 1 \tag{7.33}$$

For the $(i+1)$th bit we get:

$$a_{i+1} = c_{i+1} \oplus 1 = m(x_i, a_i, c_i) \oplus 1 = m(x_i, a_i, a_i \oplus 1) \oplus 1 = \tag{7.34}$$

$$= x_i a_i \oplus x_i(a_i \oplus 1) \oplus a_i(a_i \oplus 1) \oplus 1 = \tag{7.35}$$

$$= x_i \oplus 1 \tag{7.36}$$

So, we can easily find the value of x_i for each i. When $i = 31$, x_{31} can be any. For the case when $i = 0$, considering that $c_0 = 0$, from (7.33) we get:

$$a_0 = 1$$

Therefore, if $a_0 = 1$ then (7.31) is solvable in constant time. The solutions are $X = \overline{A \gg 1} + i2^{32}, i = 0, 1$. Finally, for the whole system, we have that solution exist if $a_0 = b_0 = 1$, which means with probability 2^{-2}. □

Lemma 4 *There exists an algorithm (**Al4**) for finding all the solutions for equations of the following type:*

$$(X \oplus C) + A = (X + B) \oplus K \tag{7.37}$$

Proof. We base our algorithm fully on the results of [115]. There, Paul and Preneel show, in particular, how to solve equations of the form:

$$(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma.$$

Let us XOR to the both sides of (7.37) $A \oplus B \oplus C$ and denote $\tilde{K} = K \oplus A \oplus B \oplus C$. Then, the equation gets the following form:

$$((X \oplus C) + A) \oplus A \oplus B \oplus C = (X + B) \oplus \tilde{K}.$$

For the $(i + 1)$th bit position, we have:

$$\tilde{k}_{i+1} = s_{i+1} \oplus t_{i+1},$$

where s_i is the carry at the ith position of $(X \oplus C) + A$, and t_i is the carry at ith position of $X + B$. From the definition of s_i we get:

$$\begin{aligned} s_{i+1} =& (x_i \oplus c_i)a_i \oplus (x_i \oplus c_i)s_i \oplus a_i s_i = \\ =& (x_i \oplus c_i)a_i \oplus (x_i \oplus c_i \oplus a_i)s_i = \\ =& (x_i \oplus c_i)a_i \oplus (x_i \oplus c_i \oplus a_i)(\tilde{k}_i \oplus t_i) \end{aligned}$$

From the definition of t_i we get:

$$t_{i+1} = x_i b_i \oplus x_i t_i \oplus b_i t_i$$

This means that $\tilde{k}_{i+1}$ can be computed from x_i,a_i,b_i,c_i,t_i, and $\tilde{k}_i$.
Further, we apply the algorithm demonstrated in [115]. The only difference is that for each bit position we have only two unknowns x_i and t_i, whereas in [115] have three unknowns. Yet, this difference is not crucial, and the algorithm can be applied.
Our experimental results (Monte-Carlo with 2^{32} trials), show that the probability that a solution exists, when A, B, C and K are randomly chosen is around 2^{-12}. □

Now, we can present our algorithm for finding solutions for the system of equations. With **Al1** we find a difference R (and values for T_0^1, T_0^2) such that equation (7.8) holds. Actually, for the same difference R many different solutions (T_0^1, T_0^2) exist (experimental results show that when (7.8) is solvable then there are around 2^5 solutions). Then, we pass as an input to **Al2** the difference R and we find a difference D (and values for H_0^1, H_0^2) such that (7.9) holds. Again for a fixed R and D many (H_0^1, H_0^2) exist. Experimentally for each random R and "good" D there are around 2^{10} solutions. Using **Al3** we check if we can find solutions for (7.10), i.e. we try to find T_1 and T_{15}. Notice, the input of **Al3** are the previously found $T_0^1, T_0^2, H_0^1, H_0^2$. If **Al3** can not find a solution then we get another pair H_0^1, H_0^2 (or generate first new difference D and then generate another 2^{10} pairs H_0^1, H_0^2). If **Al3** finds a solution to (7.10), then with **Al4** we try to find solutions for (7.11),(7.12) where the input to **Al4** are already found H_0^1, H_0^2. If **Al4** can not find a solution, then we can take different pair H_0^1, H_0^2 (or generate first new difference D and then generate H_0^1, H_0^2) and then apply first **Al3** and then **Al4**.

7.2.5 Complexity of PSA1

Let us try to find the total complexity of the algorithm. We keep in mind that when analyzing the initial equations we have used the assumption $(A+B) \gg r = (A \gg r) + (B \gg r)$, which holds with probability $\frac{1}{4}$(see [46]). In total, we used this assumption 5 times. In the equation for ΔT_0 we can control the exact value of M_1, so in total we have used the assumption 4 times. Therefore the probability that a solution of the system is a solution for the initial equations is 2^{-8}. This means that we have to generate 2^8 solutions for the system. Let us find the cost for a single solution.

The average complexity for both **Al1** and **Al2** is 2^1 steps. We confirmed experimentally that, for a random difference R, there exists a solution for Equation (7.8) with the probability 2^{-27}. So this takes $2^{27} \cdot 2^1 = 2^{28}$ steps using **Al1** and it finds 2^5 solutions for Equation (7.8). Similarly, for a random difference D, there is a solution for Equation (7.9) with the probability 2^{-27}. Therefore, this consumes $2^{27} \cdot 2^1 = 2^{28}$ steps and finds 2^{10} pairs (H_0, H_0) for Equation (7.9). The probability that a pair is a good pair for Equation (7.10) is 2^{-1} and that it is a good pair for Equations (7.11) and (7.12) is 2^{-12}. Thus, we need $2^1 \cdot 2^{12} = 2^{13}$ pairs, which we can be generated in $2^{28} \cdot 2^3 = 2^{31}$ steps. Since we need 2^8 solutions, the total complexity is 2^{39}. Note that this complexity estimate (a step) is measured by the number of calls to the algorithms that solve our specific equations. If we assume that a call to the algorithms is four times less efficient than the call to the functions f or g (which on average seems to be the case), and consider the fact that the compression function makes a total of around 2^8 calls to the functions f or g, then we get that the total complexity of the collision search is around 2^{33} compression function calls.

Notice that when a solution for the system exists, then this still does not mean that we have a pseudo collision. This is partially because we cannot control some of the values directly. Indeed we can control directly only $H_0^1, H_0^2, t_0^1, t_0^2$. The rest of the variables, i.e. $T_0^1, T_0^2, T_1, T_{15}$ we will control through the message words M_i or

with the input variables H_i, where $i > 0$. Since we pass these values as arguments for the non-injective function f we may experience situation when we can not get the exact value that we need. Yet, with overwhelming probability we can find the exact values. Let us suppose we have a solution $(H_0^1, H_0^2, T_0^1, T_0^2, T_1, T_{15}, t_0^1, t_0^2)$ for the system of equations. First we find a message word M_0 such that $f(L_{15}, H_0^1, M_0, C_0) = T_0^1$. Notice that L_{15} can be previously fixed by choosing some value for H_7. Then $f(L_{15}, H_0^2, M_0, C_0) = T_0^2$. We choose M_1 such that $[M_1 + (T_0^2 \wedge C_1)] \gg 7 - [M_1 + (T_0^1 \wedge C_1)] \gg 7 = (T_0^2 \wedge C_1) \gg 7 - (T_0^1 \wedge C_1) \gg 7$. This way the probability the previous identity becomes 1. Then we find H_1 such that $f(T_0^1, H_1, M_1, C_1) = T_1$. At last, we find M_{15} such that $f(T_{14}, L_{15}, M_{15}, C_{15}) = T_{15}$. If such M_{15} does not exist, then we can change the value of T_{14} by changing M_{14} and then try to find M_{15}. An example of a colliding pair found with **PSA1** is given in Tbl. 7.1.

Table 7.1: Colliding pairs for the compression function of LAKE found with **PSA1**.

h_0	63809228	6cc286da	00000000	00000000
	00000000	00000000	00000000	00000540
h_0'	ba3f5d77	6cc286da	00000000	00000000
	00000000	00000000	00000000	00000540
M	55e07658	00000009	00000000	00000000
	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000
	00000000	00000000	00000002	5c41ab0e
t_0	0265e384	00000000		
t_1	aba71835	00000000		
S	00000000	00000000	00000000	00000000
H	79725351	e61a903f	730aace9	756be78a
	b679b09d	de58951b	f5162345	14113165

7.2.6 PSA2 on Processmessage Procedure

In **PSA2** we introduce difference only in the chaining value H_0. Hence, this difference after the saltstate procedure, will produce differences in L_0 and L_8. In the first application of the processmessage procedure the following differential is used (Fig. 4):

1. Let L_0 have some specially chosen difference. Also, L_8 has some difference that depends on the difference in L_0.

2. After the application of the function f only $T_0, T_1, \ldots, T_8$ have non-zero differences

3. After the application of the function g all N_i have zero differences

Again, we should prove that this differential trail is possible. Basically, we should check only for the updates with non zero input differences and zero output difference (other updates hold trivially). Hence, we should prove that we can get zero difference in T_9 and $N_i, i = 0, \ldots, 8$. Since f is non-injective it is possible to get a zero difference in T_9. For $N_0, \ldots, N_8$ is also possible to get zero differences because their updating functions g always have at least two arguments with differences (check Fig. 4). Therefore, this differential is valid.

Now, let us write the system of equations that we require. Note that $T_i^1 - T_i^2 = \delta_i, i = 0, \ldots, 8$.

$$f(L_{15}, L_0^j, M_0, C_0) = \quad T_0^j, \quad j = 1,2 \tag{7.38}$$
$$f(T_0^j, L_1, M_1, C_1) = \quad T_1^j, \quad j = 1,2 \tag{7.39}$$
$$f(T_{i-1}^j, L_i, M_i, C_i) = \quad T_i^j, \quad i = 2, \ldots, 6, j = 1,2 \tag{7.40}$$
$$f(T_7, L_8^j, M_8, C_9) = \quad T_8^j, \quad j = 1,2 \tag{7.41}$$
$$f(T_8^j, L_9, M_9, C_9) = \quad T_9, \quad j = 1,2 \tag{7.42}$$
$$g(T_{15}, T_0^j, L_0^j, T_1^j) = \quad N_0, \quad j = 1,2 \tag{7.43}$$
$$g(N_{i-1}, T_i^j, L_i, T_{i+1}^j) = \quad N_i, \quad i = 1, \ldots, 7, j = 1,2 \tag{7.44}$$
$$g(N_7, T_8^1, L_8^1, T_9) = \quad g(N_7, T_8^2, L_8^2, T_9) \tag{7.45}$$

Let us focus on (7.44). This equation can be rewritten as:

$$(N_{i-1} + T_i^1) \ggg 1 \oplus (L_i + T_{i+1}^1) = (N_{i-1} + T_i^2) \ggg 1 \oplus (L_i + T_{i+1}^2) \quad (= N_i)$$

Similarly, as in **PSA1**, we get the following equation:

$$((X + A) \oplus X) \ggg 1 = (Y + B) \oplus Y, \tag{7.46}$$

where $X = N_{i-1} + T_i^2, A = T_i^1 - T_i^2, Y = L_i + T_{i+1}^2, B = T_{i+1}^1 - T_{i+1}^2$. In **Al3**, we have explained how to split this equation into two equations, $\overline{((X + A) \oplus X) = -1}$,$\overline{(Y + B) \oplus Y = -1}$, and solve them separately. The solution $X = \overline{A \gg 1}, Y = \overline{B \gg 1}$ exists when LSB of A and B are 1. Hence, for N_{i-1} and L_i we get:

$$N_{i-1} = \overline{(T_i^1 - T_i^2) \gg 1} - T_i^2 = \overline{\delta_i \gg 1} - T_i^2 \tag{7.47}$$
$$L_i = \overline{(T_{i+1}^1 - T_{i+1}^2) \gg 1} - T_{i+1}^2 = \overline{\delta_{i+1} \gg 1} - T_{i+1}^2 \tag{7.48}$$

If we put these values in the equation for N_i we get:

$$N_i = (N_{i-1} + T_i^2) \ggg 1 \oplus (L_i + T_{i+1}^2) = \overline{\delta_i \gg 1} \ggg 1 \oplus \overline{\delta_{i+1} \gg 1} \tag{7.49}$$

This means that we can split the equations of type (7.44) into two equations and solve them separately. Also, from (7.47) and (7.48) we get that $N_i = L_i$.

Now let us explain how to get two pairs that satisfy the whole differential. First, by choosing randomly $L_0^1, L_0^2, L_{15}, M_0, L_1$, and M_1, we produce a solution for the

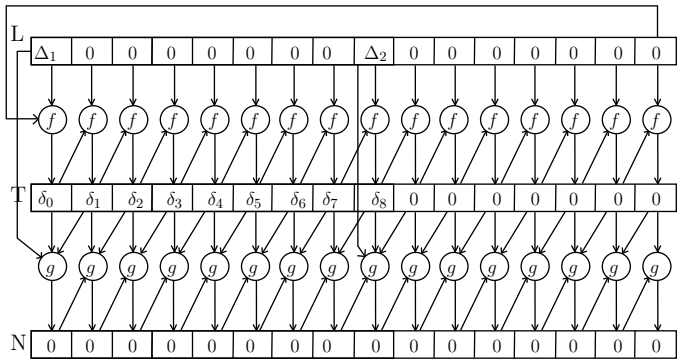

Figure 7.4: Differences in the variables for the PSA2 on processmessage. Some of the arrows that come from words with zero differences are not shown.

equations (7.38),(7.39), (7.43) and the split (7.44). Actually, we need to satisfy only the split (7.44), i.e. $N_0 = \overline{(T_1^1 - T_1^2) \gg 1} - T_1^2 = \overline{\delta_1 \gg 1} - T_1^2$, because the values of $T_0^j, T_1^j, j = 1,2$ can be any, and finding a solution for (7.43) is trivial. Then, by taking some M_2 and L_2 we produces $T_2^j = f(T_1^j, L_2, M_2, C_2), j = 1,2$. Having the values of δ_1 and δ_2 we can find the new value of L_1:

$$L_1 = N_1 = \overline{\delta_1 \gg 1} \ggg 1 \oplus \overline{\delta_2 \gg 1}$$

Since we have changed the value of L_1, then the values of T_1^1 and T_1^2 might change. Therefore, we find another value of M_1 such that the old values of T_1^1, T_1^2 stay the same. Note, that is is not always possible. Yet, with probability 2^{-2} this value can be found. As a result, we have fixed the values of M_1, L_1, T_2^1, and T_2^2. Using the same technique, we can fix the values of $M_2, \ldots, M_6, L_2, L_6, T_3^j, T_7^j, j = 1,2$ such that (7.44) would hold for $i = 2, \ldots, 6$. In short, the following is done. Let the values of N_{i-1}, M_i, L_i, T_i^1, and T_i^2 be fixed. First we generate any T_{i+1}^1 and T_{i+1}^2. Then we find the value of L_i from (7.48). Then, we change the value of M_i. This way, the values of T_i^1, T_i^2 stay the same, but now $N_{i+1}, T_i^j, M_i, L_i, T_{i+1}^j, j = 1,2$ satisfy (7.44).

Now let us fix the right T_8^1, T_8^2 such that:

$$f(T_8^1, L_9, M_9, C_9) = f(T_8^2, L_9, M_9, C_9) \tag{7.50}$$

We try different M_8, S_0 (notice that the values of L_8^1, L_8^2 depend on L_0^1, L_0^2, and S_0), and create different pairs (T_8^1, T_8^2). If this pair satisfies (7.50) and (7.47) then we change M_7 and L_7 as described previously. Finally, we change M_9 and L_9 so that (7.45) will hold. First, we find the good value of T_9 from the equation $T_9 = \overline{\Delta_2 \gg 1} - L_8^2$ and than change M_9 and L_9 to achieve this value. As a result, we have fixed all the values such that all equations hold.

7.2.7 PSA2 on Full Compression Function and Its Complexity

After the Processmessage procedure, there are no differences in any of the state variables. The feedword procedure, which produces the new chaining value, depends on the initial chaining value, the internal state variables, the salt, and the block index. Since there is difference only in the initial chaining value (only in H_0), it means that there has to be a difference in the new chaining value H_0 (and only there). If we repeat the **PSA2** on processmessage with different input difference Δ_1, we can produce a near collision with a low Hamming difference. Also, if we attack the truncated digest LAKE-224, with first 32 bits being truncated, we can find a real collisions for the compression function of LAKE-224.

Now, let us estimate the complexity of **PSA2**. For finding good random L_0^1, L_0^2, L_{15}, M_0, L_1, and M_1 that satisfies the first set of equations we have to try 2^{32} different values. For successfully fixing the correct $L_i, M_i, i = 1, \ldots, 7$, we have to start with $(2^2)^7 = 2^{14}$ different δ_1. For finding a good pair (T_8^1, T_8^2) that satisfies (7.50) and (7.47) we have to try $2^{27} \cdot 2^{32} = 2^{59}$ different M_8, S_8. Hence, the total attack complexity is around 2^{105} computations. If we apply the same reasoning for computing the complexity in the number of compression function calls (instead of computations) as it was done in the previous attack, we will get that the near collision algorithm requires around 2^{99} calls to the compression function of LAKE-256. An advanced technique that successfully finds the values of M_i, can reduce the complexity by factor of 2^{14}, hence the attack complexity will drop to 2^{85} calls.

Chapter 8

Boomerang Attacks on BLAKE-32

The SHA-3 competition [105] will soon enter the third and final phase, by selecting 5 out of 14 second round candidates. The hash function BLAKE [8] is among these 14 candidates, and it is one of the few functions that has not been tweaked from the initial submission in 2008. Being an addition-rotation-xor (ARX) design, BLAKE is one of the fastest functions on various platforms in software. Indeed, among the fastest candidates, BLAKE has the highest published security level, i.e. the best published attacks work only on a small fraction of the total number of rounds. Few attacks, however, were published on the round-reduced compression function and keyed permutation of BLAKE-32 (which has 10 rounds). In [67] Ji and Liangyu present collision and preimage attacks on 2.5 rounds of the compression function of BLAKE-32. Su et al. [133] give a near collisions on 4 rounds with a complexity of 2^{21} compression function calls. However, one can argue that the message modification they use, requires an additional effort of 2^{64} (see Sec. 8.3). Aumasson et al. [7], among other, present near collisions on 4 rounds of the compression function with 2^{56} complexity, and impossible differentials on 5 rounds of the keyed permutation.

We show various boomerang distinguishers on round-reduced BLAKE-32. Our analysis is based on the fact that BLAKE-32, being a keyed permutation, has some high probability differential trails on two, three and four rounds. First, we use these trails to build boomerang distinguishers for the round-reduced keyed permutation of BLAKE-32 on up to 8 rounds. Then we extend the concept of boomerang distinguishers to compression functions. As far as we know, this is the first application of the standard boomerangs to compression function. An amplified boomerang attack applied to hash functions was presented in [69], however it was used in addition to a collision attack. Our boomerang attacks, on the other hand, are standalone distinguishers, and work in the same way as for block ciphers – by producing the quartet of plaintexts and ciphertexts (input chaining values and output chaining values). We also show how to obtain a simpler zero-sum distinguisher from the

boomerang and present such distinguishers for 4, 5, 6 rounds of BLAKE-32. Our final result is a boomerang distinguisher for 7 rounds of the compression function of BLAKE-32. The summary of our results is given in Tbl. 8.

Although in this paper we focus on BLAKE-32, our attacks can be easily extended to the other versions of BLAKE. The attacks do not contradict any security claims of BLAKE.

Table 8.1: Summary of the attacks on the compression function (CF) and the keyed permutation (KP) of BLAKE-32.

Attack	CF/KP	Rounds	CF/KP calls	Reference
Free-start collisions	CF	2.5	2^{112}	[67]
Near collisions[a]	CF	4	2^{21}	[133]
Near collisions	CF	4	2^{56}	[7]
Impossible diffs.	KP	5	-	[7]
Boomerang dist.	CF	4	2^{67}	Sec. 8.3
Boomerang dist.	CF	5	$2^{71.2}$	Sec. 8.3
Boomerang dist.	CF	6	2^{102}	Sec. 8.3
Boomerang dist.	CF	6.5	2^{184}	Sec. 8.3
Boomerang dist.	CF	7	2^{232}	Sec. 8.3
Boomerang dist.	KP	4	2^{3}	Sec. 8.4
Boomerang dist.	KP	5	$2^{7.2}$	Sec. 8.4
Boomerang dist.	KP	6	$2^{11.75}$	Sec. 8.4
Boomerang dist.	KP	7	2^{122}	Sec. 8.4
Boomerang dist.	KP	8	2^{206}	Sec. 8.4

[a] The attack assumes that message modification can be used anywhere in the trail.

8.1 Description of BLAKE32

The compression function of BLAKE-32 processes a state of 16 32-bit words represented as 4×4 matrix. Each word in BLAKE-32 has 32 bits. In the *Initialization* procedure, the state is loaded with a chaining value $h_0, \ldots, h_7$, a salt $s_0, \ldots, s_3$, constants $c_0, \ldots, c_7$, a counter t_0, t_1 as follows:

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \longleftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix}$$

After the *Initialization*, the compression function takes 16 message words $m_0, \ldots, m_{15}$ as inputs and iterates 10 rounds. Each round is composed of eight applications of the $\mathtt{G}$ function. A column step:

$$\mathtt{G}_0(v_0, v_4, v_8, v_{12}), \mathtt{G}_1(v_0, v_4, v_8, v_{12}), \mathtt{G}_2(v_0, v_4, v_8, v_{12}), \mathtt{G}_3(v_0, v_4, v_8, v_{12})$$

followed by the diagonal step:

$$\mathtt{G}_4(v_0, v_5, v_{10}, v_{15}), \mathtt{G}_5(v_1, v_6, v_{11}, v_{12}), \mathtt{G}_6(v_2, v_7, v_8, v_{13}), \mathtt{G}_7(v_3, v_4, v_9, v_{14})$$

$\mathtt{G}_i, i \in \{0, \ldots, 7\}$ depend on their indices , message words $m_0, \ldots, m_7$, constants $c_0, \ldots, c_{15}$ and round index r. At round r, $\mathtt{G}_i(a, b, c, d)$ is described with following steps:

$$\begin{aligned}
&1 : a \leftarrow a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}) \\
&2 : d \leftarrow (d \oplus a) \ggg 16 \\
&3 : c \leftarrow c + d \\
&4 : b \leftarrow (b \oplus c) \ggg 12 \\
&5 : a \leftarrow a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\
&6 : d \leftarrow (d \oplus a) \ggg 8 \\
&7 : c \leftarrow c + d \\
&8 : b \leftarrow (b \oplus c) \ggg 7,
\end{aligned}$$

where σ_r belongs to the set of permutations as specified in [8]. The *Finalization* procedure in BLAKE-32 is depicted as:

$$\begin{aligned}
h_0^{'} &\leftarrow h_0 \oplus s_0 \oplus v_0 \oplus v_8 \\
h_1^{'} &\leftarrow h_1 \oplus s_1 \oplus v_1 \oplus v_9 \\
h_2^{'} &\leftarrow h_2 \oplus s_2 \oplus v_2 \oplus v_{10} \\
h_3^{'} &\leftarrow h_3 \oplus s_3 \oplus v_3 \oplus v_{11} \\
h_4^{'} &\leftarrow h_4 \oplus s_0 \oplus v_4 \oplus v_{12} \\
h_5^{'} &\leftarrow h_5 \oplus s_1 \oplus v_5 \oplus v_{13} \\
h_6^{'} &\leftarrow h_6 \oplus s_2 \oplus v_6 \oplus v_{14} \\
h_7^{'} &\leftarrow h_7 \oplus s_3 \oplus v_7 \oplus v_{15},
\end{aligned}$$

where $h_0, \ldots, h_7$ is the initial chaining value and $v_0, \ldots, v_{15}$ is the state value after ten rounds, and $h'_0, \ldots, h'_7$ are the words of the new chaining value.

8.2 Boomerang Attacks on Block Ciphers and Compression Functions

The boomerang attack [139] is a differential-type attack that exploits high probability differential trails in each half of a cipher E. When successful, it outputs a quartet of plaintexts and corresponding ciphertexts with some fixed particular differences between some of the pairs. This property can be used to distinguish the cipher from a random permutation, and in some cases, to recover the key.

Let us decompose the initial cipher E into two ciphers E_0, E_1, i.e. $E = E_1 \circ E_0$. Let $\Delta \rightarrow \Delta^*$ be some differential trail for E_0 that holds with probability p and $\nabla \rightarrow \nabla^*$ be a trail for E_1 with probability q. We start with a pair of plaintexts $(P_1, P_2) = (P_1, P_1 \oplus \Delta)$ and produce a pair of corresponding ciphertexts $(C_1, C_2) = (E(P_1), E(P_2))$. Then we produce a new pair of ciphertexts $(C_3, C_4) = (C_1 \oplus \nabla^*, C_2 \oplus \nabla^*)$, decrypt this pair, and get the corresponding pair of plaintexts

$(P_3, P_4) = (E^{-1}(C_3), E^{-1}(C_4))$. The difference $P_3 \oplus P_4$ is Δ with probability at least p^2q^2: 1) the difference $E_0(P_1) \oplus E_0(P_2)$ is Δ^* with probability p; 2) the differences $E_1^{-1}(C_1) \oplus E_1^{-1}(C_3)$, $E_1^{-1}(C_2) \oplus E_1^{-1}(C_3)$ are both ∇ with probability q^2; 3) when 1), 2) hold, then the difference $E_1^{-1}(C_3) \oplus E_1^{-1}(C_4)$ is Δ^* (with probability pq^2) and $E^{-1}(C_3) \oplus E^{-1}(C_4)$ is Δ with probability p^2q^2.

We would like to address a couple of issues. First, the boomerang distinguisher can be used even in the case when it returns a pair (P_3, P_4) with a difference $P_3 \oplus P_4$ specified only in certain bits (instead of the full plaintext). When the difference is specified in t bits ($t < n$), then the probability of the boomerang (in order to be used as a distinguisher) should be higher than 2^{-t}, i.e. $p^2q^2 > 2^{-t}$. Second, the real probability of the boomerang is $\hat{p}^2\hat{q}^2$, where $\hat{p}, \hat{q}$ are so-called amplified probabilities, defined as:

$$\hat{p} = \sqrt{\sum_{\Delta^*} P[\Delta \to \Delta^*]^2}, \hat{q} = \sqrt{\sum_{\nabla} P[\nabla \to \nabla^*]^2}. \tag{8.1}$$

Since finding these values is hard, in some cases, we try to get experimental results for the probability of the boomerang. We run a computer simulation, start the boomerang with a number of pairs with some prefixed difference Δ, and count the number of returned pairs that have the same difference Δ. Obviously the ratio of the returned pairs to the launched pairs is the probability of the boomerang.

The main obstacle for applying the boomerang attack to compression functions, is that in general, the compression functions are non-invertible. Hence, after obtaining the pairs (C_3, C_4) from (C_1, C_2), one cannot go backwards and obtain the pair (P_3, P_4). One way to deal with this is to switch to amplified boomerang attacks [72]. However, this type of boomerang requires internal collisions, hence in the case when the underlying compression functions are double pipes, the attack complexity becomes too high.

Indeed, the standard boomerang attack can be used as a differential distinguisher for a compression function F. The idea is to start the attack in the middle of F and then go forward and backwards to obtain the quartets, thus escaping the feedforward. Let $F(H)$ be obtained from some invertible function $f(H)$ with a feedforward, for example Davies-Meyer mode $F(H) = f(H) \oplus H$[1]. As in the attack on block ciphers, the first step is to decompose f into two functions f_0, f_1 and to find two differential trails for f_0 and f_1 (further we use the same notation as in the attacks on block ciphers). We start with four states S_1, S_2, S_3, S_4 at the end of the function f_0 (beginning of f_1) such that $S_1 \oplus S_2 = S_3 \oplus S_4 = \Delta^*$ and $S_1 \oplus S_3 = S_2 \oplus S_4 = \nabla$. From these states we obtain the initial states (input chaining values) P_i[2] and the final states (output chaining values without the feedforward) C_i, i.e. $P_i = f_0^{-1}(S_i), C_i = f_1(S_i), i = 1, \ldots, 4$. Then with probability at least p^2q^2 we have:

$$P_1 \oplus P_2 = \Delta, \qquad P_3 \oplus P_4 = \Delta$$
$$C_1 \oplus C_3 = \nabla^*, \qquad C_2 \oplus C_4 = \nabla^*.$$

[1] Indeed, $F(H, M) = E_M(H) \oplus H$, further we use $f(H) = E_M(H)$.

[2] The inputs P_i are vectors, i.e. $P_i = (H_i, M_i)$.

Extending the following attack to the whole compression function F is trivial – we just have to take into account that $C_i = f(P_i) = F(P_i) \oplus P_i$. For the boomerang quartet (P_1, P_2, P_3, P_4) we get:

$$P_1 \oplus P_2 = \Delta, \qquad P_3 \oplus P_4 = \Delta \tag{8.2}$$

$$[F(P_1) \oplus P_1] \oplus [F(P_3) \oplus P_3] = \nabla^*, \quad [F(P_2) \oplus P_2] \oplus [F(P_4) \oplus P_4] = \nabla^* \tag{8.3}$$

For a random n-bit compression function F, the complexity of finding a quartet (P_1, P_2, P_3, P_4) with the above relations (8.2),(8.3), is around[3] 2^n . Hence when $p^2q^2 > 2^{-n}$ one can launch a boomerang attack and thus obtain a distinguisher for F. The distinguisher becomes even more powerful if the attacker finds several boomerang quartets with the same differences Δ, ∇^*.

A zero-sum distinguisher, can be obtained based on the boomerangs. If in (8.3), we XOR the two equations, we get:

$$\begin{aligned}
0 =& [F(P_1) \oplus P_1] \oplus [F(P_3) \oplus P_3] \oplus \nabla^* \oplus [F(P_2) \oplus P_2] \oplus [F(P_4) \oplus P_4] \oplus \nabla^* = \\
=& F(P_1) \oplus F(P_2) \oplus F(P_3) \oplus F(P_4) \oplus (P_1 \oplus P_2) \oplus (P_3 \oplus P_4) = \\
=& F(P_1) \oplus F(P_2) \oplus F(P_3) \oplus F(P_4) \oplus \Delta \oplus \Delta = \\
=& F(P_1) \oplus F(P_2) \oplus F(P_3) \oplus F(P_4)
\end{aligned}$$

Finding a zero-sum distinguisher for a random permutation requires $2^{n/4}$ encryptions. However, since we have the additional conditions on the plaintexts (the XORs of the pairs are fixed), the complexity rises to $2^{n/2}$.

It is important to notice that to produce the quartet (for the boomerang or the zero-sum boomerang) one has to start not necessarily from the middle states (S_1, S_2, S_3, S_4). For example, one can start from two input chaining values $(P_1, P_2) = (P_1, P_1 \oplus \Delta)$, produce the values $(S_1, S_2) = (f_0(P_1), f_0(P_2))$, then obtain the values for the two other middle states $(S_3, S_4) = (S_1 \oplus \nabla, S_2 \oplus \nabla)$, and finally get the two input chaining values $(P_3, P_4) = (f_0^{-1}(S_3), f_0^{-1}(S_4))$ and the four output chaining values $(f_1(S_1) \oplus P_1, f_1(S_2) \oplus P_2, f_1(S_3) \oplus P_3, f_1(S_4) \oplus P_4)$. Clearly, the probability of the boomerang stays the same. Starting from the beginning (or from some other particular state before the feedforward) can be beneficial in the cases when one wants to use message modification or wants to have some specific values in one of the four states (as shown further in the case of BLAKE-32).

Round-reduced Differential Trails in BLAKE-32

Our attacks are based on the following round-reduced differential trails for BLAKE-32.

Observation 1 *A 2-round differential trail can be obtained in BLAKE-32 with probability* 2^{-1}.

[3] This holds only when the difference between the messages is fixed as well. Otherwise, the complexity is only $2^{n/2}$.

Table 8.2: Differential trails used in the Boomerang Attack on 4 rounds of BLAKE-32. On the left is the top trail, while on the right is the bottom trail of the boomerang. ΔM is the message difference, while ΔV_i are the differences in the state. In the left trail (top trail), ΔV_0 is the starting difference of the trail, i.e. $\Delta V_0 = \Delta$, and ΔV_2 is the ending difference, i.e. $\Delta V_2 = \Delta^*$. In the right trail (bottom trail), ΔV_2 is the starting difference of the trail, i.e. $\Delta V_2 = \nabla$, and ΔV_4 is the ending difference, i.e. $\Delta V_4 = \nabla^*$. The numbers 0,1,2, and 2,3,4, indicate the rounds covered by the boomerang – the top trail starts at round 0 and ends after round 1, while the bottom trail starts at round 2 and ends after round 3.

	Δm					Δm			
	00000000	00000000	80000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		80000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
$R.$	ΔV_i				$R.$	ΔV_i			
0	00000000	80000000	00000000	00000000	2	80000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		80000000	00000000	00000000	00000000
	1					1			
1	00000000	00000000	00000000	00000000	3	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	2^{-1}					2^{-1}			
2	00000000	80000000	00000000	00000000	4	00000000	00000000	00000000	80000000
	00000000	00000000	00010000	00000000		00010000	00000000	00000000	00000000
	00000000	00000000	00000000	00800000		00000000	00800000	00000000	00000000
	00800000	00000000	00000000	00000000		00000000	00000000	00800000	00000000

Observation 2 *A 3-round differential trail may be obtained from the above described two round differential trail with probability* 2^{-s}*, where* $s = 6, 7$ *or* 8

The probability of the trails is proven in [32].

8.3 Boomerang Attacks on the Compression Function of BLAKE-32

The high probability round-reduced differential trails in the permutation of BLAKE-32 can be used to attack the compression function and find boomerang distinguishers. However, due to the *Initialization* procedure, there are a few requirements on the trails. First, since the block index is copied twice, the initial differences in v_{12} and v_{13}, as well as the differences in v_{14} and v_{15}, have to be the same. Second, even in the case when the attacker has a trail with initial differences consistent to the above requirement, if he uses message modification techniques in the higher rounds of the trail, he might end up with inconsistent initial states. For example, if the attacker uses some k-round trail and starts fixing the values of the state and the

Table 8.3: Differential trails used in the Boomerang Attack on 5 rounds of BLAKE-32.

	Δm					Δm			
	00000000	00000000	40000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	80000000	00000000	00000000
$R.$	ΔV_i				$R.$	ΔV_i			
0	00000000	40000000	00000000	00000000	2	00000800	80008000	80000000	80000000
	00000000	00000000	00000000	00000000		80000800	80008000	00000000	00000000
	00000000	00000000	00000000	00000000		80000000	80808080	80000000	00000000
	00000000	00000000	00000000	00000000		80000000	00800080	80008000	80000000
	2^{-1}					2^{-7}			
1	00000000	00000000	00000000	00000000	3	00000000	00000000	80000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	2^{-2}					1			
2	00000000	40000000	00000000	00000000	4	00000000	00000000	00000000	00000000
	00000000	00000000	00008000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00400000		00000000	00000000	00000000	00000000
	00400000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
						2^{-1}			
					5	00000000	00000000	00000000	80000000
						00010000	00000000	00000000	00000000
						00000000	00800000	00000000	00000000
						00000000	00000000	00800000	00000000

messages at round k, and then goes backward, he can obtain two states with some predefined difference (as the one predicted by the trail). However, the probability that these two states are consistent with the *Initialization* procedure is 2^{-64} (if $v_{12} \oplus v_{13} = c_4 \oplus c_5$ and $v_{14} \oplus v_{15} = c_6 \oplus c_7$). Note that if one of the states is consistent, then the other one is consistent as well (if the attacker used trails with appropriate initial difference). Therefore, using message modification techniques in later steps of the trail is not trivial (without increasing the complexity of the attack). On the other hand, the modification can still be used at the beginning because the attacker starts with two states consistent with the *Initialization* procedure.

For the boomerang attack on 4 rounds of the compression function of BLAKE-32 we can use two trails each on 2 rounds (see Tbl. 8.2). Since the probability of these trails is 2^{-1}, the probability of the boomerang is 2^{-4}. To create a quartet of states, consistent with the *Initialization* procedure, we start with a pair of states (P_1, P_2) that have a difference Δ (note that Δ does not have a difference in the "block index" words) and consistent with the *Initialization* words $v_{12}, v_{13}, v_{14}, v_{15}$ in both of the states, then go two rounds forward and obtain the pair (S_1, S_2). Then we produce the pair $(S_3, S_4) = (S_1 \oplus \nabla, S_2 \oplus \nabla)$ and go backwards two rounds to get the pair of initial states (P_3, P_4). The probability that P_3 (and therefore P_4) is consistent with the *Initialization* is 2^{-64}. Also, from S_1, S_2, S_3, S_4 we go forward two rounds, produce the outputs and apply the *Finalization* to get the new chaining values. Note that *Finalization* is linear, hence the differential trail (with XOR difference)

Table 8.4: Differential trails used in the Boomerang Attack on 6 rounds of CF of BLAKE-32.

	Δm					Δm			
	00080008	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	80000000	00000000
$R.$	ΔV_i				$R.$	ΔV_i			
4	80088008	00000000	00080008	00000000	7	80008000	00000000	00000000	00000800
	80088008	00000000	00000000	00000000		80008000	00000000	00000000	80000800
	00080008	00000000	00080008	00000000		80808080	80000000	00000000	80000000
	00000000	00000000	00000000	00000000		00800080	00008000	00000000	80000000
	2^{-21}					2^{-6}			
5	00000000	00000000	00080008	00000000	8	00000000	80000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	2^{-2}					1			
6	00000000	00000000	00000000	00000000	9	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	2^{-11}					2^{-1}			
7	00880088	00000000	00000000	00000000		00000000	80000000	00000000	00000000
	00000000	11011101	00000000	00000000		00000000	00000000	00010000	00000000
	00000000	00000000	80088008	00000000		00000000	00000000	00000000	00800000
	00000000	00000000	00000000	80008000		00800000	00000000	00000000	00000000

holds with probability 1. Therefore, we can produce the boomerang quartet with a complexity of $4 \cdot 2^{4+64} = 2^{70}$ calls to the 4-round reduced compression function of BLAKE-32.

The boomerang attack on 5 rounds is rather similar. We only need one of the trails to be on 3 rounds, instead of 2 (see Tbl. 8.3). Such a trail has a probability of 2^{-8}, hence the boomerang has a probability of 2^{-18} and the whole attack (taking into account the *Initialization*) has a complexity of around $4 \cdot 2^{18+64} = 2^{84}$ compression function calls.

For the boomerang attack on 6 rounds we will use two 3-round trails (see Tbl. 8.4). However, we cannot use the optimal trails (the ones that hold with around 2^{-7}) because the starting difference in each such trail is inconsistent with the *Initialization* procedure. Therefore, for the top trail of the boomerang we will use a trail which has lower probability 2^{-34} but has no differences in any of the "block index" words $(v_{12}, v_{13}, v_{14}, v_{15})$. For the bottom trail we can use an optimal trail. The complexity of this boomerang distinguisher on 6 rounds becomes $4 \cdot 2^{2 \cdot 34+2 \cdot 7+64} = 2^{148}$ calls.

Note, for the bottom trail for 5 and the top trail for 6 round boomerangs (see Tbl. 8.3,8.4), we did not use the best trails with probability 2^{-7}, 2^{-21}, but instead used trails with lower probability $(2^{-8}, 2^{-34})$. We found that if we use the best trails, then the boomerang does not work, most likely because of the slow diffusion. We cannot get four states in the middle (after the third round), that have pairwise

Table 8.5: Differential trails used in the Boomerang Attack on 6.5 and 7 rounds of CF of BLAKE-32.

	Δm					Δm			
	00000000	00000000	00000000	00000000		00000000	00000000	80000000	00000000
	80008000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
$R.$	ΔV_i				$R.$	ΔV_i			
0.5	00000000	80008000	00000000	00000000	3.5	00800880	c8088848	80440044	00008000
	00000000	00000000	00000000	00000000		80000000	80800880	488c0888	80040804
	00000000	00000000	00000000	80008000		00000800	00008080	80808080	00000000
	00000000	00000000	00000000	00000000		80048040	08408840	00800000	80000000
	2^{-3}					2^{-43}			
1	00000000	80008000	00000000	00000000	4	80000000	00000000	80000800	80008000
	00000000	00000000	00000000	00000000		00000000	00000000	80000800	80008000
	00000000	00000000	00000000	00000000		80000000	00000000	80000000	80808080
	00000000	00000000	00000000	00000000		80008000	00000000	80000000	00800080
	2^{-1}					2^{-6}			
2	00000000	00000000	00000000	00000000	5	80000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	2^{-3}					1			
3	00000000	00000000	00000000	80008000	6	00000000	00000000	00000000	00000000
	00010001	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00800080	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00800080	00000000		00000000	00000000	00000000	00000000
	2^{-33}					2^{-1}			
3.5	00010001	08000800	08000800	80088008	7	00000000	00000000	80000000	00000000
	02000200	10111011	11111111	11101110		00000000	00000000	00000000	00010000
	00010001	00880088	80888088	88008800		00800000	00000000	00000000	00000000
	00000000	00080008	80088008	08000800		00000000	00800000	00000000	00000000
						2^{-24}			
					7.5	00000800	08000000	80000008	00110010
						10010010	01101001	10110101	22222022
						00800008	80080080	08808080	11001101
						00000008	80080000	08800080	11001100

Δ^* and ∇ difference (Δ^* is the end difference of the top trail). However, if we take other trails, as the ones we have taken, the boomerang quartet can be obtained – we confirmed this experimentally, by producing a boomerang quartet.

Each of the above attacks can be improved if we take into account the amplified probabilities for the boomerang attack and if we use message modification. We can obtain the amplified probabilities (and the total probabilities) of the boomerang experimentally: we start with a number of plaintext pairs with the required difference Δ, and then check how many of the returned (by the boomerang) differences are Δ. Also, in the first round, for one side of the boomerang we use message modification, i.e. we pass this round with probability 1. Using these two approaches, we got the following results: the boomerang on 4 rounds has a probability 2^{-1}, on 5 rounds $2^{-5.2}$, and on 6 rounds 2^{-36}. Hence, the attack complexity for 4 rounds drops to $4 \cdot 2^{1+64} = 2^{67}$, for 5 rounds to $4 \cdot 2^{5.2+64} = 2^{71.2}$, and for 6 rounds to $4 \cdot 2^{36+64} = 2^{102}$ compression function calls. An example of boomerang quartet for 6 rounds, with the first pair of plaintext consistent to the *Initialization*, while only the difference in the second is consistent, and therefore obtained with around $4 \cdot 2^{36}$ compression function calls, is given in Tbl. 8.9. The complexities of the boomerang distinguishers for 4,5, and 6 round are bellow 2^{128}, therefore they can be used as zero-sum boomerang distinguishers, i.e. $P_1 \oplus P_2 = P_3 \oplus P_4 = \Delta$ and $F(P_1) \oplus F(P_2) \oplus F(P_3) \oplus F(P_4) = 0$.

For the boomerang on 6.5 rounds, we use a top trail on 3 rounds (from 0.5 to 3.5) with 2^{-40}, and a bottom trail on 3.5 rounds (from 3.5 to 7), with 2^{-50} (see Tbl. 8.5). The complexity of producing the boomerang quartet is $4 \cdot 2^{2 \cdot 40 + 2 \cdot 50 + 64} = 2^{246}$ compression function calls. The probability of the first round in the top trail is 2^{-3}, hence using message modification does not lower significantly the attack complexity. However, computing the amplified probabilities can improve the attack. Obviously, we cannot do this experimentally, as the probability of the boomerang is too low – $2^{-2 \cdot 40 - 2 \cdot 50} = 2^{-180}$. Therefore, we cannot test for the whole 6.5 rounds, but we can do it for a reduced number of rounds. We tested for only half round at the end of the first trail (round 3 to round 3.5). We start with a pair of states with a difference specified by the top trail at round 3 and go half round forward to obtain a new pair of states. Then, to each element of the pair, we XOR the same difference (the one specified by the bottom trail at round 3.5), and produce a new pair states. Finally, we go backwards a half round, and check if the difference in the pair is at the one we have started with. Note that the half round can be split into four G functions, and for each of them the amplified probabilities can be found independently. By doing so, we found that the amplified probability for this half round of the boomerang is 2^{-26} instead of twice 2^{-33}, i.e. $2^{-2 \cdot 33} = 2^{-66}$. Another low probability part of the boomerang is the top half round of the second trail – round 3.5 to round 4 holds with 2^{-43}. In this part we can use message modification. We start at round 3.5 with four states that have pairwise differences Δ^* and ∇. We go half round forward and obtain four states with pairwise differences as specified by the bottom trail at round 4. To obtain such states we need $4 \cdot 2^{2 \cdot 43} = 2^{88}$. Once we have this half round boomerang, we can freely change the message words that are not taken as inputs in this half round without altering the input and the output values of the half round. Hence, we have $2^{8 \cdot 32} = 2^{256}$ degrees of

freedom. From the middle states we can obtain the initial and final states (and the chaining values). Therefore, the total complexity of the boomerang on 6.5 rounds becomes $2^{88} + 4 \cdot 2^{2\cdot(3+1+3)+26+2\cdot(6+1)+128} = 2^{184}$ calls. Note that unlike as in the case of the boomerangs on 4 and 5 rounds, now the probability that the initial states are consistent to the *Initialization*is 2^{-128} because we use message modification in the middle rather than in the beginning. The bottom trail can easily be extended for additional half round (see Tbl. 8.5) with probability 2^{-24}. Therefore, the boomerang on 7 rounds requires around $2^{184+2\cdot24} = 2^{232}$ compression function calls.

8.4 Boomerang Attacks on the Keyed Permutation of BLAKE-32

Further we present boomerang attacks on the keyed permutation of BLAKE-32, assuming that the key is unknown to the attacker. These attacks can be seen as distinguishers for the internal cipher of BLAKE-32. The cipher takes 512-bit plaintexts and 512-bit key, and after 10 rounds, outputs 512-bit ciphertext (we discard the *Initialization*and *Finalization* procedures). Switching from the boomerangs for

Table 8.6: Differential trails used in the Boomerang Attack on 6 rounds of KP of BLAKE-32.

	Δm					Δm			
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	80000000	00000000	00000000	00000000		00000000	80000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
$R.$	ΔV_i				$R.$	ΔV_i			
0	80008000	80000000	80000000	00000800	3	80008000	00000000	00000000	00000800
	80008000	00000000	00000000	80000800		80008000	00000000	00000000	80000800
	80808080	80000000	00000000	80000000		80808080	00000000	00000000	80000000
	00800080	80008000	00000000	80000000		00800080	00000000	00000000	80000000
	2^{-6}					2^{-6}			
1	00000000	80000000	00000000	00000000	4	00000000	80000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	1					1			
2	00000000	00000000	00000000	00000000	5	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	2^{-1}					2^{-1}			
3	00000000	00000000	00000000	80000000	6	00000000	80000000	00000000	00000000
	00010000	00000000	00000000	00000000		00000000	00000000	00001000	00000000
	00000000	00800000	00000000	00000000		00000000	00000000	00000000	00800000
	00000000	00000000	00800000	00000000		00800000	00000000	00000000	00000000

the compression function to the boomerangs for the keyed permutation has advantages and disadvantages for the attacker. On one hand, the attacker is not con-

cern any more about the *Initialization* procedure, and he can use any trails for the boomerang. On the other hand, since the key is unknown, he cannot use message modification techniques to improve the probability of the boomerang.

The boomerangs on 4 and 5 rounds of the keyed permutation of BLAKE-32 have the same probability as in the case of compression function: 2^{-4} for 4 rounds, and 2^{-18} for 5 rounds. For 6 rounds, we can use two high probability trails ($2^{-7}, 2^{-7}$, see Tbl. 8.6), and therefore, the probability of the boomerang is 2^{-28}. If we take into account the amplified probabilities, and fix the returning difference only in 128 bits (the words v_1, v_5, v_9, v_{13}) instead of in 512 bits, for the total complexity of the boomerang attack we get 2^3 encryptions for 4 rounds, $2^{7.2}$ for 5 rounds, and $2^{11.75}$ for 6 rounds. These results were confirmed on a PC and a boomerang quartet for 6 rounds is presented in Tbl. 8.8.

Table 8.7: Differential trails used in the Boomerang Attack on 7 and 8 rounds of KP of BLAKE-32.

	Δm					Δm			
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	80000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	80000000	00000000	00000000		00000000	00000000	00000000	00000000
$R.$	ΔV_i				$R.$	ΔV_i			
1.5	80440044	00008000	80800880	48088848	5.5	80808000	80888080	c80c8008	80440044
	488c0888	00040804	80000000	80800880		80040804	80800000	80888000	c8880088
	80808080	80000000	00000880	00008080		80000000	00000800	00808080	80000080
	00800000	80000000	00040040	88c00840		00800000	00048000	08408840	80800000
	2^{-42}					2^{-44}			
2	00000800	80008000	80000008	80000000	6	00008000	80000000	00000000	80000800
	80000800	80008000	00000000	00000000		80008000	00000000	00000000	00000800
	80000000	80808080	80000000	00000000		00808080	80000000	00000000	00000080
	80000000	00800080	80008000	80000000		80808080	80008000	00000000	80800000
	2^{-6}					2^{-7}			
3	00000000	00000000	80000000	00000000	7	00000000	80000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	1					1			
4	00000000	00000000	00000000	00000000	8	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
	2^{-1}					2^{-1}			
5	00000000	00000000	00000000	80000000	9	00000000	80000000	00000000	00000000
	00010000	00000000	00000000	00000000		00000000	00000000	00010000	00000000
	00000000	00800000	00000000	00000000		00000000	00000000	00000000	00800000
	00000000	00000000	00800000	00000000		00800000	00000000	00000000	00000000
	2^{-24}					2^{-30}			
5.5	00110010	00000800	08000000	80000008	9.5	08000000	80000008	80110018	00000800
	22222022	10010010	01101001	10110101		01101001	10110101	32332123	10010010
	11001101	00800008	80080080	08808080		80080080	08808080	19809181	00800008
	11001100	00000008	80080000	00880080		80080000	08800080	19801180	00000008

The boomerangs for 7 and 8 rounds, are rather similar: for 7 rounds we use two trails on 3.5 rounds (the first from round 2 to round 5.5, and the second from

round 5.5 to round 9), and for 8 rounds, we just extend these trails for additional half round (see Tbl. 8.7). The complexity of the boomerangs is $4 \cdot 2^{2 \cdot 31 + 2 \cdot 52} = 2^{168}$ for 7 rounds and $4 \cdot 2^{2 \cdot 55 + 2 \cdot 82} = 2^{276}$ for 8 rounds. Again, as in the case of 6.5-round boomerang on the compression function, we can compute experimentally the lower bounds on the amplified probabilities, by only testing the probability of the first half round of the bottom trail. We get 2^{-48} instead of $2^{-2 \cdot 44}$. Also, we can fix the returning difference only in 256 bits, instead of 512 bits, and thus increase the probability in the first half round of the top trail by a factor of 2^{-6} for 7 rounds, and 2^{-30} for 8 rounds. Hence, the boomerang on 7 rounds requires at most 2^{122}, and on 8 rounds at most 2^{206} encryptions.

Examples of Boomerang quartets

Table 8.8: Example of a boomerang quartet for 6 round-reduced keyed permutation of BLAKE-32.

P_1	7d8a1f02	206849ad	42413a50	d702fa14	facc9c67	11306e7c	eba852eb	4f31f62f
	993e3958	bc426fcc	55033261	b2ac26a9	6dfc2edd	32163c44	ef989577	2d6d6bb4
P_2	fd8a9f02	a06849ad	c2413a50	d702f214	7acc1c67	11306e7c	eba852eb	cf31fe2f
	19beb9d8	3c426fcc	55033261	32ac26a9	6d7c2e5d	b216bc44	ef989577	ad6d6bb4
P_3	de971194	ae012c6a	4422f8ea	fff2d41b	80a79b50	b1d61b36	fe8c23fe	a883faf9
	e1dab487	e4971af1	51dbf40b	6e32fb27	7c797796	19b156e9	16e0ac52	a12eefcb
P_4	5e979194	2e012c6a	c422f8ea	fff2dc1b	00a71b50	b1d61b36	fe8c23fe	2883f2f9
	615a3407	64971af1	51dbf40b	ee32fb27	7cf97716	99b1d6e9	16e0ac52	212eefcb
$P_1 \oplus P_2$	80008000	80000000	80000000	00000800	80008000	00000000	00000000	80000800
	80808080	80000000	00000000	80000000	00800080	80008000	00000000	80000000
$P_3 \oplus P_4$	80008000	80000000	80000000	00000800	80008000	00000000	00000000	80000800
	80808080	80000000	00000000	80000000	00800080	80008000	00000000	80000000
M_1	a0a28e67	1fd77849	83d86d19	4a72bc82	3704f04d	bb57c994	37612239	0f7ad68a
	df14386d	4e2e05c7	55d1a87f	187d8225	fcc527c5	96071c3e	4ae251d8	52de23f2
M_2	a0a28e67	1fd77849	83d86d19	4a72bc82	b704f04d	bb57c994	37612239	0f7ad68a
	df14386d	4e2e05c7	55d1a87f	187d8225	fcc527c5	96071c3e	4ae251d8	52de23f2
M_3	a0a28e67	1fd77849	83d86d19	4a72bc82	3704f04d	3b57c994	37612239	0f7ad68a
	df14386d	4e2e05c7	55d1a87f	187d8225	fcc527c5	96071c3e	4ae251d8	52de23f2
M_4	a0a28e67	1fd77849	83d86d19	4a72bc82	b704f04d	3b57c994	37612239	0f7ad68a
	df14386d	4e2e05c7	55d1a87f	187d8225	fcc527c5	96071c3e	4ae251d8	52de23f2
$M_1 \oplus M_2$	00000000	00000000	00000000	00000000	80000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
$M_1 \oplus M_3$	00000000	00000000	00000000	00000000	00000000	80000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
$M_2 \oplus M_4$	00000000	00000000	00000000	00000000	00000000	80000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
C_1	928c1f77	3aa097f2	4d5589bb	f307e618	c8ea4ebc	c63769df	64e2b7ba	f2c76b2b
	c909808a	672bcdf3	260608d6	7de7ba36	749c4e7d	aef2defd	b7d3318a	5080389e
C_2	9948791c	21c19a0f	8804efac	d56588e4	c6f6b101	32456224	20c423d5	df0105fe
	33ee8883	23bde21d	bedb2451	2c673c2f	bf7d194d	cfc78321	5ec259f9	a9c8786b
C_3	928c1f77	baa097f2	4d5589bb	f307e618	c8ea4ebc	c63769df	64e3b7ba	f2c76b2b
	c909808a	672bcdf3	260608d6	7d67ba36	741c4e7d	aef2defd	b7d3318a	5080389e
C_4	9948791c	a1c19a0f	8804efac	d56588e4	c6f6b101	32456224	20c523d5	df0105fe
	33ee8883	23bde21d	bedb2451	2ce73c2f	bffd194d	cfc78321	5ec259f9	a9c8786b
$C_1 \oplus C_3$	00000000	80000000	00000000	00000000	00000000	00000000	00010000	00000000
	00000000	00000000	00000000	00800000	00800000	00000000	00000000	00000000
$C_2 \oplus C_4$	00000000	80000000	00000000	00000000	00000000	00000000	00010000	00000000
	00000000	00000000	00000000	00800000	00800000	00000000	00000000	00000000

Table 8.9: Example of a boomerang quartet for 6 round-reduced compression function of BLAKE-32. Note that the initial states P_1, P_2 are consistent with the *Initialization*.

P_1	30841585	41abc330	447466d0	17ae8472	b94fc56d	e9cb678a	1d9d6e9e	eb558123
	66d322c2	23cbae19	52e9bb2a	dd6b8f2b	ea1cd197	678ad865	6594bdd4	81f42bc5
P_2	b08c958d	41abc330	447c66d8	17ae8472	39474565	e9cb678a	1d9d6e9e	eb558123
	66db22ca	23cbae19	52e1bb22	dd6b8f2b	ea1cd197	678ad865	6594bdd4	81f42bc5
P_3	f3383666	710fc071	1990f347	34475dd7	7d41ddc9	68e231ed	ea9bba79	a4990860
	d7ede8b5	f1c0b054	1c754989	a0e95ceb	3d259f5f	878bffae	f511b0fd	def26a26
P_4	7330b66e	710fc071	1998f34f	34475dd7	fd495dc1	68e231ed	ea9bba79	a4990860
	d7e5e8bd	f1c0b054	1c7d4981	a0e95ceb	3d259f5f	878bffae	f511b0fd	def26a26
$P_1 \oplus P_2$	80088008	00000000	00080008	00000000	80088008	00000000	00000000	00000000
	00080008	00000000	00080008	00000000	00000000	00000000	00000000	00000000
$P_3 \oplus P_4$	80088008	00000000	00080008	00000000	80088008	00000000	00000000	00000000
	00080008	00000000	00080008	00000000	00000000	00000000	00000000	00000000
M_1	7670ae70	c6539713	373c66b6	3d4522c3	b66689d0	37ee4f5d	467de620	9aabd357
	b6b3b13c	c6d41a4c	cb994b4c	b79e16fa	8a9d8079	9914ccb1	9c68b051	86d41e1e
M_2	7678ae78	c6539713	373c66b6	3d4522c3	b66689d0	37ee4f5d	467de620	9aabd357
	b6b3b13c	c6d41a4c	cb994b4c	b79e16fa	8a9d8079	9914ccb1	9c68b051	86d41e1e
M_3	7670ae70	c6539713	373c66b6	3d4522c3	b66689d0	37ee4f5d	467de620	9aabd357
	b6b3b13c	c6d41a4c	cb994b4c	b79e16fa	8a9d8079	9914ccb1	1c68b051	86d41e1e
M_4	7678ae78	c6539713	373c66b6	3d4522c3	b66689d0	37ee4f5d	467de620	9aabd357
	b6b3b13c	c6d41a4c	cb994b4c	b79e16fa	8a9d8079	9914ccb1	1c68b051	86d41e1e
$M_1 \oplus M_2$	00080008	00000000	00000000	00000000	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
$M_1 \oplus M_3$	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	80000000	00000000
$M_2 \oplus M_4$	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	80000000	00000000
C_1	3f432ef6	5f89fb80	7283d8cf	13731945	344d16f8	2203b3b5	74b3637e	52ed9169
	efcea8db	32b84ffc	57cfa772	2258156c	22696ef4	53cb7ac6	3ab6294a	ce58038c
C_2	f284e034	f866e60d	1e52775f	f6f764cb	ef09e2e8	da83b2d1	a4a869d1	f22eefb0
	821c38c2	6da245e0	7b52665c	0f8ce3ba	7ed4c20c	ef76217d	77835c6d	184a17e3
C_3	3f432ef6	df89fb80	7283d8cf	13731945	344d16f8	2203b3b5	74b2637e	52ed9169
	efcea8db	32b84ffc	57cfa772	22d8156c	22e96ef4	53cb7ac6	3ab6294a	ce58038c
C_4	f284e034	7866e60d	1e52775f	f6f764cb	ef09e2e8	da83b2d1	a4a969d1	f22eefb0
	821c38c2	6da245e0	7b52665c	0f0ce3ba	7e54c20c	ef76217d	77835c6d	184a17e3
$C_1 \oplus C_3$	00000000	80000000	00000000	00000000	00000000	00000000	00010000	00000000
	00000000	00000000	00000000	00800000	00800000	00000000	00000000	00000000
$C_2 \oplus C_4$	00000000	80000000	00000000	00000000	00000000	00000000	00010000	00000000
	00000000	00000000	00000000	00800000	00800000	00000000	00000000	00000000

Part IV

Tools for Automatic Search of Related-Key Differential Characteristics in Block Ciphers

It is hard to overestimate the importance of differential attacks to the analysis of block ciphers. The complexity of these attacks is tightly related to the probability of the differentials used in the attack – the higher the probability of the differential, the lower the complexity of the attack. Although a differential is a collection of differential characteristics (trails), in practice, the attacker finds a single characteristic and uses it in the differential attacks while assuming that the probability of the differential is the same as the probability of the characteristic. Therefore, the task for the attacker is to find a high probability characteristic. Quite often these characteristics are found by hand. When the attacker considers related-key attacks, as opposite to the simple single-key attacks, then the search space increases significantly. Therefore, it seems that automatizing the search of related-key characteristics is beyond the reach.

In this part of the contribution, we show that for some large classes of block ciphers, automatic search algorithms can be built. We focus in particular on byte-oriented ciphers (such as AES) and on DES-like ciphers. Although the underlying transformations in these primitives are very distinct (the first type being byte-oriented, while the second is bit-oriented), we show that a similar approach can be used to build tools that find the best (i.e. with the highest probability) related-key differential characteristics in the ciphers. Interestingly, the task of finding the best characteristics can be reduced to the following graph problem. First the attacker builds, according to the underlying round transformation of the cipher, a directed graph G where each edge has a weight. Then the task of finding the best related-key characteristic on k rounds for the cipher is equivalent to the problem of finding a path in the graph G of length k with a minimal weight. The graph problem can be solved using several approaches:

1. *Dynamic programming.* First take all paths of length 1. If an end vertex (of a path) has a degree greater than one, keep only the paths (incident to the vertex) that have the lowest weight. Extend each such path for an additional edge and repeat the process, i.e. for each end vertex with degree greater than one, keep only the paths with lowest weight. After k iterations, all paths that are left have length k. Among all such paths, take the one that has the lowest weight – this is the path of length k with the minimal weight. The dynamic programming approach has a time complexity and memory complexity (to store the intermediate paths) linear in the number of edges of the graph G. The memory requirement is a big drawback for using this approach for our purposes – the feasibility of dynamic programming is limited to the cases where the number of edges does not exceed 2^{32}.
2. *Matsui's approach.* Given the minimal weights $w_1, \ldots, w_{k-1}$ of the paths with lengths $1, 2, \ldots k-1$, and some path with a weight w_k^* of length k, one builds the minimal weight path with length k by the following method. First, he takes a path of length 1 and checks if the weight does not exceed $w_k^* - w_{k-1}$. If it is less, then he extends the path for an additional edge, i.e. he builds a path of length 2 with a weight w_2^*. Again, he checks if $w_2^* < w_k^* - w_{k-2}$. This process is repeated until the path with length k is built. If the weight of this path is less than w_k^*, then w_k^* is updated with the weight of the built path. If this method is repeated for all paths of length 1, then it will always find the minimal weight path of length k. The weight check is introduced to cut off the unnecessary extension of some paths. For example, if a path with length t has a weight more than $w_k^* - w_{k-t}$, then to extend it to a path of length k one would need to concatenate it with a path with a length $k-t$, hence add a weight of at least w_{k-t}, and therefore the total weight of the path on all k rounds would be greater than w_k^*. The time complexity of Matsui's approach in the worst case is exponential in the number

of edges. In practice, this approach can be feasible as shown further. It is important to notice that the memory requirement is negligible (when the graph is built on the fly).

3. *The split approach.* If the path of length k, has a minimal weight w, then there is an edge in the path of a weight at most $\lceil \frac{w}{k} \rceil$. Hence, when building the path one can start with all possible edges that have a weight at most $\lceil \frac{w}{k} \rceil$, and try to add $k-1$ more edges such that the total weight of the path would not exceed w. Note that the position of the minimal edge in the path is unknown, hence one should try all possible positions. The time complexity of the split approach in the worst case is exponential in the number of edges, while the memory complexity is negligible (when the graph is built on the fly).

Further we show how to apply Matsui's and the split approach, as well as a combination of the two approaches to the search of related-key differential characteristics in the mentioned above ciphers. The results presented in this contribution were published in the following papers:

- [30] **Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others**, EUROCRYPT 2010
- [108] **Tweaking AES**, SAC 2010
- [31] **Search for Related-key Differential Characteristics in DES-like ciphers**, FSE 2011

Chapter 9

The Search in Byte-oriented Ciphers

Proving the security of modern block ciphers against differential [22] and linear cryptanalysis [92], i.e. finding the best differential and linear characteristics, has become a well understood and relatively simple task. Many of the modern ciphers are constructed as so-called substitution-permutation networks (SPN) — they consist of layers of non-linear substitution boxes (S-boxes) and diffusion layers built from linear or affine functions. The designer simply has to use diffusion layers with high (or maximal) branch number which is typically achieved by using maximum distance separable matrices [43]. Using such diffusion layers one can prove lower bounds on the number of active S-boxes for a certain number of internal rounds. The designer then picks the number of rounds for which the probability of the best differential or linear characteristic is lower than 2^{-k} where k is the key size of a cipher. The resultant cipher is then provably secure against standard differential and linear attacks.

Such reasoning however holds only in the single key model, and does not extend to the case of related-key attacks [16]. In this class of cryptanalytic attacks the attacker knows or chooses the relation between several keys and is given access to encryption/decryption functions with all these keys. The goal of the attacker is to find the actual keys. The relation between the secret keys is a function chosen by the attacker with some extra care taken to avoid trivial attacks, and quite often it is just a XOR with a chosen constant. Security of most modern block ciphers against related-key attacks still relies on heuristic and ad hoc arguments. This situation is very similar to the heuristic security that we have for the modern hash functions, which is due to a lack of proper tools and methodologies for the analysis of differentials of non-bijective functions.

We make a step in the direction of provable security of modern block ciphers (and by analogy of modern hash functions), by presenting an efficient tool that can evaluate and help to prove bounds for the security of block-ciphers (hash functions) against differential related-key (open-key or chosen message) attacks.

Automatic search for best differential characteristics and linear approximations in a single key scenario was first performed by Matsui [93] for DES. Algorithms for automatic search of differential characteristics for MD4 were presented in [128, 54], and for MD5 in [132]. De Cannière and Rechberger in [37] described a method that finds characteristics in SHA-1 in an automatic way and produced the best known collision trails for SHA-1. A typical problem that arises when trying to construct a tool for automatic search of characteristics

is the size of the search space. The search space is exponential in the size of the block and the key which makes straightforward approaches infeasible for 128-bit block 128-256 bit key ciphers[1]. Therefore often the most important task for producing an efficient tool is the reduction in the size of the internal state by using some equivalent representation of the state, but with smaller size.

In that respect it is natural to look at byte (or word)-oriented ciphers which constitute a large fraction of modern ciphers. A natural compact (sometimes known as truncated) representation would shrink each byte into a single bit, representing by 0 a byte without difference and by 1 a byte with a difference. In such a representation a 16-byte block state, 16-32 byte key would translate into 16 and 16-32 bits respectively. These numbers are low, and give hope that a search of the whole 2^{32} – 2^{48} space of related-key differential characteristics might be possible. The main problem with this representation is a very heavy branching which will happen in the linear diffusion layers and on the XORs. Such representation alone will only allow to search for the most basic and short characteristics which happen with probability close to 1.

Our goal is to perform a full search for related-key differential characteristic and to be able to find or to prove the non-existence of characteristics similar to those that were used in the recent attack on AES-256 [29]. In this paper we achieve this goal for all versions of AES and for several other ciphers. At the basis of our related-key search algorithm, further denoted as a *tool*, lies Matsui's approach for search of the best differential characteristics, with several important modifications. Depending on the key schedule of a cipher, we differentiate three classes of block ciphers. This is done to improve the efficiency. For each of the classes we introduce a special modification to Matsui's algorithm to obtain the final tool. The internal representation of the difference in a cipher (state and subkeys) plays a very important role for constructing a feasible tool. Using only compact representation may lead to a high branching (caused by XORs or other linear-diffusion transforms such as MixColumns in AES) when trying to build all possible one round characteristics. We completely eliminate the branching in the state of a cipher by using a special representation that takes into account the properties of the matrix used in the linear-diffusion layer. The related-key differential characteristics produced by our tool, fix only the positions of the active bytes[2]. To produce standard differential characteristics, i.e. characteristic with exact values of the differences in the active bytes, one has to fix the byte differences corresponding to the possible transitions of the differences trough the S-boxes.

We apply the tool to different byte-oriented block ciphers. The tool finds related-key characteristics for the full-round ciphers, or if such characteristics do not exist, for the maximal number of rounds for which they exist. We provide the best possible differential characteristics for all the versions of AES. For AES-128 it is on 5 rounds out of 10 (this also means that AES-128 is secure against straightforward related-key attacks after 6 rounds). For AES-192 it is on 9 rounds – three rounds short of the total 12 rounds. The characteristic for AES-256 is on all 14 rounds, and it is the same characteristic (and the only one on 14 rounds) that was given in [29]. Then we present boomerang attack on AES-128 reduced to 7 rounds, and improve the complexity of the best attack on AES-192 by a factor 2^7. We analyze the version of Camellia without the FL functions, and where the rotation constants in the key schedule are multiplies of 8. For this "byte-Camellia", the best related-key differential characteristic is on 8 rounds (out of 18). Additionally, we launch a chosen-key

[1]Note that full search was feasible for 64-bit block cipher DES, due to its Feistel structure, which reduces the search space to about 2^{32}.

[2]Note that while our characteristics allow certain flexibility in the values due to the compact representation of differences – they are not *truncated differentials* since our goal is to find fully specified differential characteristics, rather than just truncated characteristics.

attack and find a characteristic on all 18 rounds of byte-Camellia. For Khazad first we find a related-key characteristic on 7 rounds (out of 8), and then we show a boomerang attack on 7 rounds, and a chosen-key attack on the full-round Khazad. For the ciphers FOX and Anubis, we show that related-key differential characteristics cannot exist on more than 4-5 rounds. The summary of our results is given in Tables 9.1,9.2. Due to space limitations, we will not describe the ciphers that we analyze, we refer the reader to [43, 2, 13, 70, 12], and will use the original notation proposed by the designers in these papers.

Table 9.1: Summary of attacks on the ciphers examined in the paper

Cipher	Attack/Result	Rounds	Data	Workload	Reference
AES-128	Collisions	7	2^{32}	2^{128}	[57]
	Partial sum	7	$2^{128}-2^{119}$	2^{120}	[52]
	Impossible diff.	7	$2^{112.2}$	$2^{117.2}$	[90]
	Boomerang - RK	7	2^{97}	2^{97}	Sect. 9.2.2
AES-192	Rectangle - RK	9	2^{64}	2^{143}	[62]
	Rectangle - RK	10	2^{125}	2^{182}	[81]
	Boomerang - RK	12	2^{123}	2^{176}	[28]
	Boomerang - RK	12	2^{116}	2^{169}	Sect. 9.2.3
AES-256	Rectangle - RK	10	2^{114}	2^{173}	[20, 81]
	Subkey Diff.	10	2^{48}	2^{49}	[26]
	Differential - RK	14	2^{131}	2^{131}	[29]
	Boomerang - RK	14	$2^{99.5}$	$2^{99.5}$	[28]
Camellia-128	Impossible	11	2^{118}	2^{126}	[91]
byte-Camellia-128	Chosen-key dist.	18	$2^{6\cdot17}$	$2^{6\cdot17}$	Sect. 9.3.2
Khazad	Slide attack[a]	5	2^{98}	2^{104}	[24]
	Integral	5	2^{64}	2^{91}	[103]
	Boomerang[a] - RK	7	2^{50}	2^{50}	Sect. 9.4.2
	Chosen-key dist.	8	2^{55}	2^{55}	Sect. 9.4.3

[a] The attack works for a weak key class, and the workload includes the effort to find related keys from the class.

Table 9.2: The upper bounds on the probabilities of the related-key differential characteristics for full or round-reduced ciphers examined in the paper. The probabilities for a higher number of rounds are below 2^{-k}, where k is the key size.

Cipher	Rounds	Workload	Section
AES-128	5	$2^{-6\cdot17}$	9.2.2
AES-192	9	$2^{-6\cdot25}$	9.2.2
AES-256	14[b]	2^{-131}	9.2.2
byte-Camellia-128	8	$2^{-6\cdot19}$	9.3.1
Khazad	7	$2^{-5\cdot19}$	9.4.1

[b] The same characteristics as in [29].

9.1 A Tool for Search of Related-Key Differential Characteristics in Byte-oriented Ciphers

Related-key differentials, introduced by Biham in [16], unlike the traditional single-key differentials that have a difference only in the plaintext, have a difference in the key as well. A related-key differential is specified with two input differences: Δ_P in the plaintext and Δ_K in the key, and an output difference Δ_C in the ciphertext. A pair of plaintexts (P_1, P_2) and a pair of keys (K_1, K_2) follow the related-key differential in the cipher $E_K(P)$, if $P_1 \oplus P_2 = \Delta_P, K_1 \oplus K_2 = \Delta_K$ and $E_{K_1}(P_1) \oplus E_{K_2}(P_2) = \Delta_C$. A popular technique to find lower bounds on the probability of differentials is via finding probabilities of the best differential characteristics. A related-key differential characteristic besides the differences in the key, plaintext and ciphertext, also fixes the differences in the state and the subkeys after each round of the cipher.

There are a couple of approaches to construct a tool for search of the best round-reduced (related- or single-key) differential characteristics. One approach is using dynamic programming. Let $\Delta X_i, \Delta Y_j, \Delta Z_k$ be the differences only in the plaintext in the case of single-key, or in both the plaintext and the subkeys in case of related-key differentials. First, all one round characteristics $\Delta X_i \rightarrow \Delta Y_j$ are built, i.e. the attacker tries all possible starting differences X_i, and for each of them goes through one round of the cipher and obtains the differences Y_j. Distinct starting differences X_{i_1}, X_{i_2} can produce the same difference Y_j. For each Y_j only the characteristics, that have the highest probability are left. Next, the attacker builds again all one round characteristics $Y_j \rightarrow Z_k$, for different Y_j (but only those Y_j that were obtained in the first step). Again, for each Z_j he selects only the characteristics that have the highest probability. As a result, he had built the optimal two-round characteristics $X_i \rightarrow Y_j \rightarrow Z_k$. This procedure is repeated until the target n-round differential characteristic is built. The time complexity of the dynamic programming approach is linear in the number of one round characteristics, but it requires a lot of memory for storing the intermediate round values $\Delta Y, \Delta Z$, etc. That is why we will use the second approach, similar to the one used by Matsui in [93] for finding the best differential and linear characteristics in DES. It requires relatively small memory and the time complexity highly depends on the probability of the best round-reduced differential characteristics. The algorithm works by induction: to find the best n-round characteristic first it finds the best $1, 2, \ldots, n-1$ round characteristics. However, Matsui's approach at some stage requires building all one round characteristics – their number depends on the size of the search space. Thus a straightforward application of Matsui's search to modern ciphers would immediately fail but there is some hope for byte-oriented ciphers if one switches to compact[3] representations in which each byte is replaced by a single bit: a byte with a difference, also called an *active* byte, is replaced by 1, a byte without a difference — by 0. This means that the difference in n-byte cipher can be represented as n-bit vector.

The compact representation seems optimal, yet several improvements to Matsui's algorithm are still required. Almost all byte-oriented ciphers are designed as substitution-permutations networks (SPN), i.e. they have a layer of S-boxes (S-layer) and a linear diffusion layer – a simple multiplication of the input by a matrix A (P-layer). When the S-boxes are bijective (a property common to most ciphers developed in the last 15 years), then an active byte stays active (and vice-versa) before and after the S-box. Hence, the S-layer does not alter the compact representation. On the other hand, the P-layer can change the number of the active bytes as well as their positions (depending on the exact values of the

[3] Often, this is called a truncated representation. We use the term compact to avoid confusion with truncated differentials, since our tool finds standard differential characteristics, rather than truncated.

differences in the active bytes, and the branch number of the matrix A) and therefore it introduces a branching. Thus, besides the traditional compact representation, further denoted as S-value, we will introduce additional representation, called P-value. Indeed, the difference in n-byte cipher will be represented as $2n$-bit vector, where the first n coordinates (bits) are the S-value coordinates, and the next n are the P-value coordinates. The P-value of a difference is obtained when S-value goes through a P-layer and it is the same as the previous S-value (see Fig.9.1 for clarification). For example, in AES, if the value of a difference of some column is (0,1,0,0,0,0,0,0) (i.e. there is a difference only in the second byte of the column, the difference is of a type $(0,x,0,0)^T$) before the MixColumn, then after the MixColumn it is (0,0,0,0,0,1,0,0) meaning: $A(0,x,0,0)^T$, where A is the MixColumn matrix and x is an arbitrary non-zero byte value (i.e. it is a four-byte difference, obtained when some column with a difference only in the second byte was multiplied by the MixColumn matrix). Note that the representation can always be reduced to only S-value (although often not uniquely). For example, the above vector (0,0,0,0,0,1,0,0) can be represented as (1,1,1,1,0,0,0,0). The P-values reduce the branching as well: it is better to XOR two P-values, then to reduce them to only S-values and then XOR them. For example, if we XOR two differences (0,0,0,0,0,1,0,0) and (0,0,0,0,0,1,0,0) then the result can be (0,0,0,0,0,1,0,0) or (0,0,0,0,0,0,0,0). On the other hand, if we first reduce them to only S-values, then we will get the values (1,1,1,1,0,0,0,0) and (1,1,1,1,0,0,0,0). Obviously, XOR of these two values gives 2^4 possible outputs. In the states of the ciphers, after each transform, we will have either only non-zero S-value or only non-zero P-value of a difference (but never both), and hence we can effectively eliminate any branching in the state (the branching goes into the key). However even in such representations the search space of 128-bit block, 256-bit key cipher would be 16+32 bits, i.e. 2^{48}. Another complication is that if one would like to search for differential characteristics rather than truncated differentials one will need to pay in heavy branching at every XOR operation both in the state and in the key-schedule, which makes the search completely infeasible. Hence, depending on the key schedule, we would like to propose different variants of the tool to solve these problems:

1. The first variant is the original Matsui's approach itself. It applies to ciphers that *have minimal branching*[4] *in the key schedule, with subkeys consecutively obtained one from another*. This means that once the difference in the subkey K_i is fixed, the difference in the subkey K_{i+1} can easily and almost uniquely be determined. Let $\Delta X \rightarrow \Delta Y$ be one round differential characteristic, where ΔX is the input difference in **both the state and the subkey**, and ΔY is the output difference, and let $W(\Delta X \rightarrow \Delta Y)$ be the weight function of this characteristic – the probability cost required to produce a pair that follows the characteristic (the exact definition of W is given later). Let $W_1, W_2, \ldots, W_{n-1}$ be the weights of the best $1, 2, \ldots, (n-1)$-round characteristic found previously with the algorithm and let $\tilde{W}_n$ be the weight of some (not necessarily optimal) n-round characteristic D_n. The search for the best n-round characteristic in pseudo code is described in Alg. 13. In short, first the algorithm builds all possible one round characteristics with a weight at most $\tilde{W}_n - W_{n-1}$. This constraint is introduced to filter some of the one round characteristics: if the weight of the first round is more than $\tilde{W}_n - W_{n-1}$ then it can not be extended to an n-round characteristic because the weight of $n-1$ rounds is at least W_{n-1} so in total it will have a weight more than the previously found characteristic of weight $\tilde{W}_n$. Each of the good one round characteristics (the one that pass the filter) is extended (when possible) to n rounds by the NextRound procedure. One call of this procedure extends the characteristic by one additional round. Again, it extends only the characteristics that satisfy the weight

[4] With regards to our representation.

condition, by checking if the sum of the weights of the r and $n-r$ characteristics is not greater than the weight $\tilde{W}_n$ of the already known differential D_n.

2. The second variant of the tool is for ciphers that *have possibly high branching in the key schedule, with subkeys consecutively obtained one from another*. A good example of this type of key scheduling is the one in AES (subkey K_{i+1} is obtained from K_i in one iteration, but due to XORs in the key schedule, there is a lot of branching). If we try to apply the variant 1 of the tool to this type of ciphers we would have to build all one round characteristics (with differences in the state and the subkey). Yet, the high branching in the subkey, blows the number of characteristic out of proportion, and the search becomes infeasible. That is why we have to modify the tool for this special case of ciphers. Let ΔS_r be the difference in the state of round r after the XOR of the subkey K_r and let ΔK_r be the difference in this subkey. To add one more round to this characteristic one can proceed as follows:

   - take ΔS_r and go through all one round transformations of the state to build $\Delta\tilde{S}_{r+1}$ which is the difference in the state of round $r+1$ just before the XOR of the subkey K_{r+1}
   - take any ΔS_{r+1}
   - XOR $\Delta\tilde{S}_{r+1}$ and ΔS_{r+1} to produce ΔK_{r+1}
   - check if ΔK_{r+1} can be obtained from ΔK_r in one round.

   This way, instead of building all one round characteristics in the subkey, we only have to check if some subkey difference can be transformed to another difference in one subkey round (see Fig. 9.1). The number of these transitions that has to be checked is related to the size and branching of the state. Usually the state has smaller size than the subkey, and with the right representation it can have minimal or no branching, leading to a feasible search. Let ΔP be the plaintext difference, $\Delta S_r, \Delta K_r$ be the difference in the state and the subkey of round r, $\Delta\tilde{S}_r$ the difference in the state of round r just before the subkey XOR, $W(\Delta S_r \rightarrow \Delta\tilde{S}_{r+1})$ the probability of the characteristic $\Delta S_r \rightarrow \Delta\tilde{S}_{r+1}$. The notions of W_i are the same as in the previous variant. For the sake of clarity we assume there is no whitening key. The variant 2 of our search tool is described in Alg. 2.

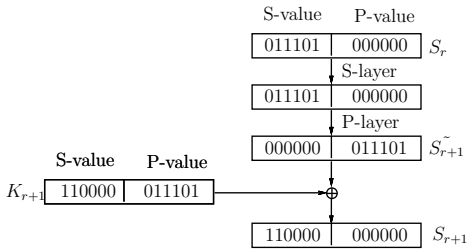

Figure 9.1: The variant 2 of the tool with S- and P-value representations

3. The third variant of the tool applies to ciphers that *have key schedule with subkeys that are not successively obtained one from another*. Usually, the key schedule of these

ciphers applies heavy transformations to the master key to obtain another key, and then combines these two keys (often with linear transformations) to get all subkeys. To build the tool we will use the following strategy: 1) from the master key, obtain all the subkeys, and 2) apply the first variant of the tool – build the characteristics for the state, but use the obtained subkeys (instead of building characteristics for subkeys). Let $\Delta X \xrightarrow{\Delta K} \Delta Y$ denote one round characteristic where $\Delta X, \Delta Y$ are the differences in the initial and final states, and ΔK is the subkey used in that round. The rest of the notions are the one used in variant 1. The pseudo code of the third variant is given in Alg. 3.

Algorithm 1 Search of n-round differential characteristics - Variant 1

for all $\{\Delta X \rightarrow \Delta Y | W(\Delta X \rightarrow \Delta Y) + W_{n-1} \leq \tilde{W}_n\}$ **do**
  Call NextRound(ΔY, $W(\Delta X \rightarrow \Delta Y)$, 2)
end for

NextRound(ΔY, w, r)
for all $\{\Delta Z | \Delta Y \rightarrow \Delta Z$ and $W(\Delta Y \rightarrow \Delta Z) + w + W_{n-r} \leq \tilde{W}_n\}$ **do**
  if $r = n$ **then**
    Update D_n
    $\tilde{W}_n \leftarrow w + W(\Delta Y \rightarrow \Delta Z)$
  else
    Call NextRound($\Delta Z, w + W(\Delta Y \rightarrow \Delta Z), r + 1$)
  end if
end for

Now, let us determine the weight function $W(\Delta_1 \rightarrow \Delta_2)$ of the one round characteristics $\Delta_1 \rightarrow \Delta_2$. In the attacks on AES [29, 28], the attacker pays only for the active S-boxes (active bytes that go through S-boxes) in the state and the subkey in each round of the cipher. Hence, we will use the same definition: $W(\Delta_1 \rightarrow \Delta_2)$ is defined as the number of active S-boxes in the state and the subkey in the one round characteristic $\Delta_1 \rightarrow \Delta_2$.

When searching for n-round differential characteristic, the upper bounds on the weight of these characteristics are limited by the maximal number of active S-boxes that a characteristic can have. These upper bounds depend on the key size and the difference propagation probability of the S-boxes which is usually 2^{-7} (sometimes 2^{-6} or even 2^{-5}). The weight of the n-round differential characteristic for a cipher with k-bit key, and S-boxes with maximal difference propagation probability 2^{-l} is upper bounded by $\lfloor \frac{k}{l} \rfloor$. The related-key differential characteristics produced by the tool have fixed positions of the active bytes, while the exact values are undefined. To produce standard differential characteristics, one has to find the exact values of the active bytes (the differences in the active bytes). The probability of the standard characteristics may be lower than the one predicted by the tool, but never higher, because the tool assumed that all active S-boxes hold with maximal differential probability, while in practice (in the case of standard characteristic) some S-boxes may hold with lower probability.

A new class of attacks, presented in [85, 29], called *open-key* attacks, gives the attacker the full freedom of *knowing* or even *choosing* the key. In return, the attacker has to demonstrate some non-trivial property of the cipher which differentiates it from an ideal cipher. The motivation behind these attacks is that ciphers are often used as building blocks for

Algorithm 2 Search of n-round differential characteristics - Variant 2

for all $\Delta P, \Delta S_1$ **do**
  Obtain $\Delta\tilde{S_1}$ from ΔP
  $\Delta K_1 = \Delta\tilde{S_1} \oplus \Delta S_1$
  Call NextRound$(\Delta S_1, \Delta K_1, W(\Delta P \rightarrow \Delta\tilde{S_1}), 2)$
end for

NextRound$(\Delta S_{r-1}, \Delta K_{r-1}, w, r)$
Obtain $\Delta\tilde{S_r}$ from ΔS_{r-1}
if $W(\Delta S_{r-1} \rightarrow \Delta\tilde{S_r}) + w + W_{n-r} \leq \tilde{W_n}$ **then**
  for all ΔS_r **do**
    $\Delta K_r = \Delta\tilde{S_r} \oplus \Delta S_r$
    if $r = n$ **then**
      Update D_n
      $\tilde{W_n} \leftarrow w + W(\Delta S_{r-1} \rightarrow \Delta\tilde{S_r})$
    else
      Call NextRound$(\Delta S_r, \Delta K_r, w + W(\Delta S_{r-1} \rightarrow \Delta\tilde{S_r}), r+1)$
    end if
  end for
end if

Algorithm 3 Search of n-round differential characteristics - Variant 3

for all master $\Delta K|$ obtain subkeys $\Delta K_1, \ldots \Delta K_n$ with weight W_K **do**
  for all $\{\Delta X \xrightarrow{\Delta K_1} \Delta Y | W(\Delta X \xrightarrow{\Delta K} \Delta Y) + W_K + W_{n-1} \leq \tilde{W_n}\}$ **do**
    Call NextRound$(\Delta Y,\ W(\Delta X \xrightarrow{\Delta K} \Delta Y),\ 2)$
  end for
end for

NextRound$(\Delta Y,\ w,\ r)$
for all $\{\Delta Z | \Delta Y \xrightarrow{\Delta K_r} \Delta Z$ and $W(\Delta Y \xrightarrow{\Delta K_r} \Delta Z) + w + W_K + W_{n-r} \leq \tilde{W_n}\}$ **do**
  if $r = n$ **then**
    Update D_n
    $\tilde{W_n} \leftarrow w + W(\Delta Y \xrightarrow{\Delta K_r} \Delta Z) + W_K$
  else
    Call NextRound$(\Delta Z, w + W(\Delta Y \rightarrow \Delta Z), r+1)$
  end if
end for

some other cryptographic primitives, such as hash functions. There, the attacker has a full freedom of choosing all input parameters. An interesting approach is applicable to all ciphers in the chosen-key attack model. We call this approach *divide-and-conquer* technique. Let us have some related-key differential characteristic for a cipher. Since we control both the key and the state (it is a chosen-key attack), we can find a good pair of keys and states that fol-

low the characteristic by the following method: 1) first find a good pair of keys that follow the differential characteristic only in the key, 2) once the subkeys are fixed, find a good pair of plaintexts that follow the differential characteristic in the state. It means we can split the whole characteristic in two halves: the one in the key, and the one in the state, and *instead of multiplying their probabilities, we can add them*. We will launch chosen related-key differential attacks on the full-round ciphers, in the cases when (secret) related-key characteristics do not exist. Note that proving the resistance against the chosen related-key differential attacks is still an open problem because it is unclear how to estimate the upper bound on the weighs of these characteristics since the number of rounds that can be covered for free varies from 1 in the rebound attack [98], 2 in the Super-Sbox [58, 87], and even more in the tool of Khovratovich et al [74].

9.2 AES

The 128-bit block version of Rijndael[43] has been standardized by NIST as Advanced Encryption Standard (AES) in November 2001 [106]. It supports three different key sizes: 128, 192, and 256 bits, denoted as AES-128, AES-192, and AES-256, respectively. Various cryptanalytic results were published on AES, and until recently, the best attacks presented non-random properties of 7/10/10 rounds (out of 10/12/14 rounds) of AES-128/192/256 [52, 57, 85, 62, 81]. A breakthrough in analysis of AES have been the results [29, 28]. In [29] a related-key attack on all 14 rounds of AES-256 was presented. In [28], boomerang attacks on full-round AES-192 and AES-256 were shown.

AES is an SPN cipher. The subkeys are generated consecutively one from another, but there is a lot of branching caused by the XORs of columns in the key schedule. Hence, *we will use variant 2 of the tool*. The state goes through four transformations: S-box layer, ShiftRows, linear-diffusion layer called MixColumns, and XOR of the key. In the tool we will use the following optimal representation of the state trough one round: the beginning state (before the S-boxes) can have non-zero only S-value (but zero P-value), after the S-box layer and after ShiftRows has again only non-zero S-value, after the MixColumns has non-zero only P-value, and after the subkey XOR again it has only non-zero S-value. This way, there is no branching in the state. The subkeys then can be determined as a XOR of a state of only P-value (the one after MixColumns) and a state of only S-value (the next round state, just before the S-boxes), hence they have columns that can have both non-zero S- and P-values. To use variant 2 of the tool we would have to be able to determine if the difference in the subkey K_{i+1} can be obtained from the difference in K_i. One subkey round consists of XOR of columns, application of S-boxes, and rotation of a column. If we represent the columns of the subkeys simply with only S-value, all of the above transforms can be easily checked. Therefore, each column of the subkeys is reduced only to S-value: 1) convert P-value into S-value, 2)XOR the obtained S-value with the initial S-value. Note, reduction to S-value as well as the XOR introduce branching, but the search is still feasible.

9.2.1 Best Characteristics for AES

We have applied the tool to all three versions of AES. The maximal difference propagation of the S-box in AES is 2^{-6}. Since the key sizes are 128,192, and 256, we can allow no more than 21, 31, and 42 active S-boxes in the characteristics for AES-128, AES-192, and AES-256, respectively. For AES-128 we found differentials characteristics on 4 rounds with 13 active S-boxes, and on 5 rounds with 17 active S-boxes. *In AES-128 there are no 6-round*

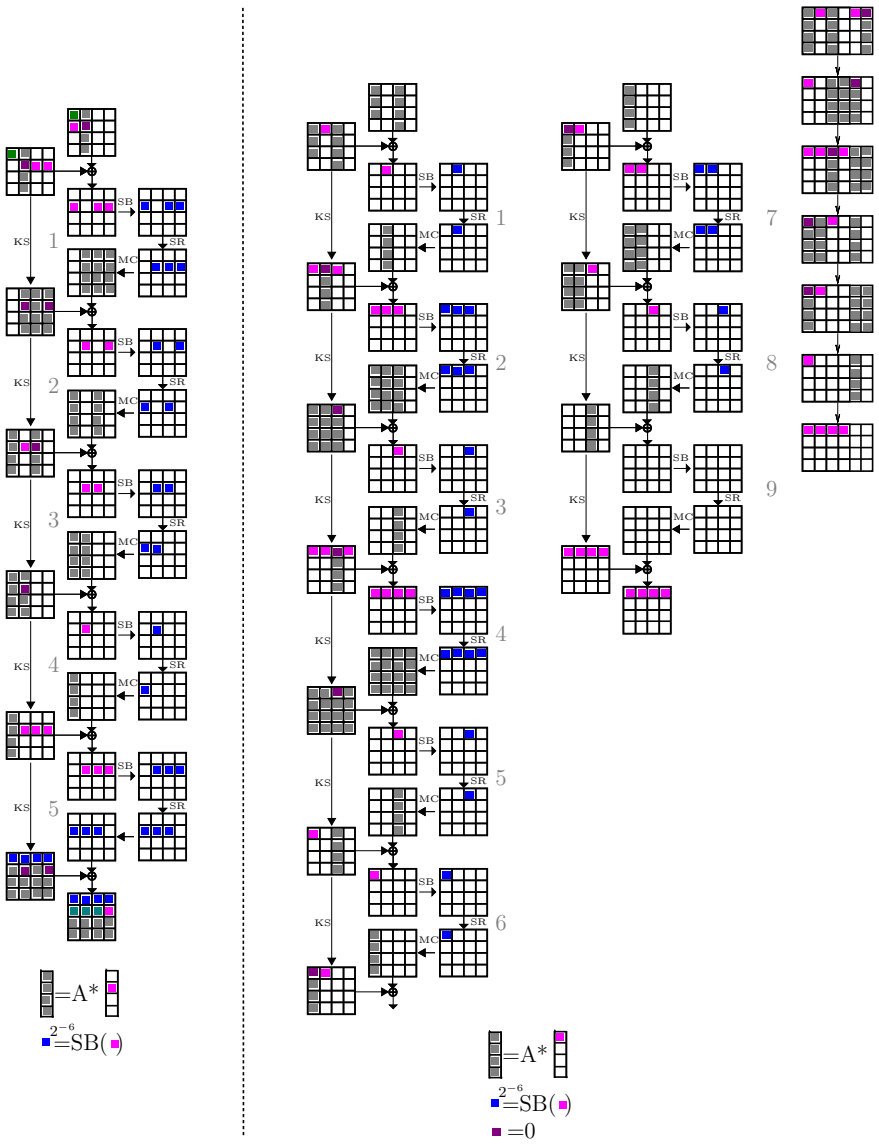

Figure 9.2: The best characteristics for AES-128(left) and AES-192(right). The first one is on 5 rounds, while the second one is on 9 rounds.

related-key differential characteristics. For AES-192 we found that the best related-key differential characteristics is on 9 rounds out of 12. The characteristic on 9 rounds has 25 active S-boxes (14 in the state, and 11 in the key). For AES-256 we found unique differential characteristic on all 14 rounds, but this is the same characteristic that was presented in [29]. The characteristic from [29] is optimal for 9-14 rounds, but we have found better characteristic on 8 rounds (10 active S-boxes instead of 14). The 5-round differential characteristic for AES-128, and 9-round for AES-192 are presented in Fig. 9.2. Regarding chosen-key attacks, in all versions of AES, there are no differential characteristic on 10 rounds with 21 or less active S-boxes in the state.

9.2.2 Related-key Boomerang Attack on 7-round AES-128

Let us show a boomerang attack on 7 rounds of AES-128. We will use two 3-round differential characteristics: a top 3-round truncated differential characteristic (4-1-4-16) with no key difference, and a 3-round related-key bottom differential characteristic with 5 active S-boxes in the state and 1 in the subkeys. They are presented in Fig. 9.3. Note that if we extend the bottom characteristic for one additional round then the difference in the ciphertexts is fixed in 9 bytes, 4 bytes have equal difference and 3 bytes have a random difference. The difference δ between these two ciphertexts can have $2^{4\cdot 7} = 2^{28}$ distinct values (1-bit of freedom is lost for each of the 4 active S-boxes, since given a fixed input difference only 2^7 output differences are possible). Let Δ be the difference between K^4 and let the key schedule transform this difference into Δ' in K^7. Instead of guessing 2^{28} possibilities of the bottom difference for each ciphertext (which would increase the pressure on our filters) we guess 31 bit of the key: seven bits of $k^6_{1,3}$ and full $k^7_{1,1}, k^7_{1,2}, k^7_{1,3}$. This guess allows us to work on both faces of the bottom characteristic, since unlike in most related-key boomerang attacks our boomerang has only two related keys, instead of four.

The attack works as follows: For each guess of 2^{31} bits of the key

1. Prepare a structure of plaintexts P_i with all the possible 2^{32} four byte values on the main diagonal and the other bytes fixed.
2. Encrypt all the plaintexts P_i with the secret key K and obtain ciphertexts C_i.
3. For each ciphertexts C_i compute the correct difference δ using the 31-bit key guess, and obtain $D_i = C_i \oplus \delta$.
4. Decrypt all D_i with the key which is computed from the last subkey: $K^7 \oplus \Delta'$ and obtain plaintexts Q_i.
5. Sort all Q_i by 12 non-diagonal bytes. Pick only the pairs (Q_i, Q_j) that have zero difference in these 12-bytes. If none are found then goto 1.
6. Check the candidate quartet against 8 active S-boxes at the top (four on both sides of the boomerang) which gives an 8-bit filter.
7. Do the key counting step with the remaining quartet candidates.

Let us calculate the data and time requirements of the attack. A pair of plaintexts passes the first round with a probability 2^{-22} (MixColumns from four to one active byte, the position and the value of the active byte is irrelevant). The next two rounds are passed with probability 1, so in the third round with have a pair of states with all bytes active. In the second characteristic, from the bottom up assume that the initial 31-bit guess was correct, then in the next three rounds we have five active S-boxes which hold with probability 2^{-30} (when each is 2^{-6}). Yet for the two pairs of ciphertexts we only need the same difference after the fourth round (from bottom up) and therefore the two S-boxes of this round can be counted

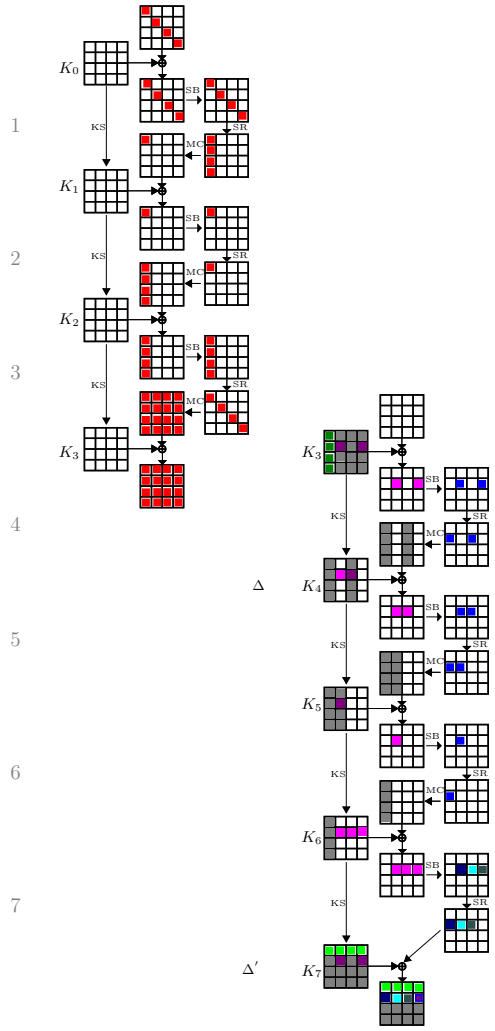

Figure 9.3: Two characteristics for the related-key boomerang attack on AES-128 reduced to seven rounds.

only once (on the one side of the boomerang the pair passes the layer of S-boxes with one of the 2^{14} possible differences, on the parallel side of the boomerang the pair matches this difference with probability 2^{-14}). So the two pairs of ciphertexts pass the second characteristic with a probability of $2^{-(3\cdot6+3\cdot6+2\cdot7)} = 2^{-50}$. Now that we have passed with low cost the 4th round layer of S-boxes where the top-down characteristic had 16 active S-boxes we switch to the last phase of the boomerang attack, where the effect of the mixing of the third round can be undone for free because are guaranteed to have the same difference as in the forward direction. In the second round we pay again 2^{-24} for four to one active byte of MixColumns(2^{-22} if we do not require the boomerang to return in exactly the same 4 bytes). The next round is done for free so we obtain two plaintexts with a difference only in four diagonal bytes. Hence the total probability of the boomerang is $2^{-22-50-24} = 2^{-96}$. Each structure of 2^{32} plaintexts contains 2^{63} pairs with a difference in the four diagonal bytes. Hence, to find two good boomerang quartets we need $2^{96-63+1} = 2^{34}$ structures or $2^{34+32} = 2^{66}$ chosen plaintexts and $2^{31} \cdot 2^{66} = 2^{97}$ adaptive chosen ciphertexts. The average amount of false quartets for all 2^{31} key guesses which satisfy our 96+8 = 104-bit filtering condition is $2^{31+34+63-104} = 2^{24}$. Note that each boomerang quartet suggest 31-bit value for the key guess at the bottom as well as 16 guesses for 64 bits at the top (corresponding to 4 active S-boxes in the plaintext at each side). Since we requested two good boomerang quartets they will vote together for the correct keys while the remaining 2^{24} false quartets would vote randomly. We expect that none of the false quartets survive this 91-bit key voting step.

At this point the attacker can either finish the attack with an exhaustive search of about 2^{96} steps or by repeating the boomerang attack starting from another 4 active S-boxes in the plaintext.

9.2.3 Related-key Boomerang Attacks on AES-192 and AES-256

We tweaked our tool to produce the optimal differential characteristics for a boomerang attack on AES-192. The tool produced a top differential characteristic other than the one presented in [28], with the same bottom characteristic. The two characteristics are shown in Fig.9.4. The ladder switch between the two characteristics in round 6 is simpler: due to the switch there are no active S-boxes in this round. The top characteristic has 2 active in round 3, and 1 in round 4, while the bottom characteristic has 1 active in round 7, 8, and 10, and 2 active in round 9. Hence, the probability of the boomerang is $2^{-6\cdot(2+1+1+1+2+1)} = 2^{-48}$ compared to the boomerang in [28] with a probability 2^{-55}. A rough estimate between these two attacks gives us a speed-up of 2^7: the new boomerang attack requires 2^{116} data, and 2^{169} time.

For AES-256, on 6 and 7 rounds there are only two characteristics with 5 active S-boxes (and no characteristics with less active S-boxes), and these are the exact characteristics used in the boomerang attack on AES-256 in [28]. On the other hand, 8-round characteristic has at least 10 active S-boxes, hence using it in a boomerang attack will blow up the complexity above the best known attack. Therefore, we believe that the characteristics used for the attack on AES-256 in [28] are optimal.

9.3 Camellia

Camellia [2] is a 128-bit SPN block cipher with 128, 192, and 256-bit keys. We will analyze Camellia with 128-bit keys, without the FL functions. This version has 18 rounds, and so far, the best cryptanalytical results are truncated differential of 8 rounds [89], and impossible

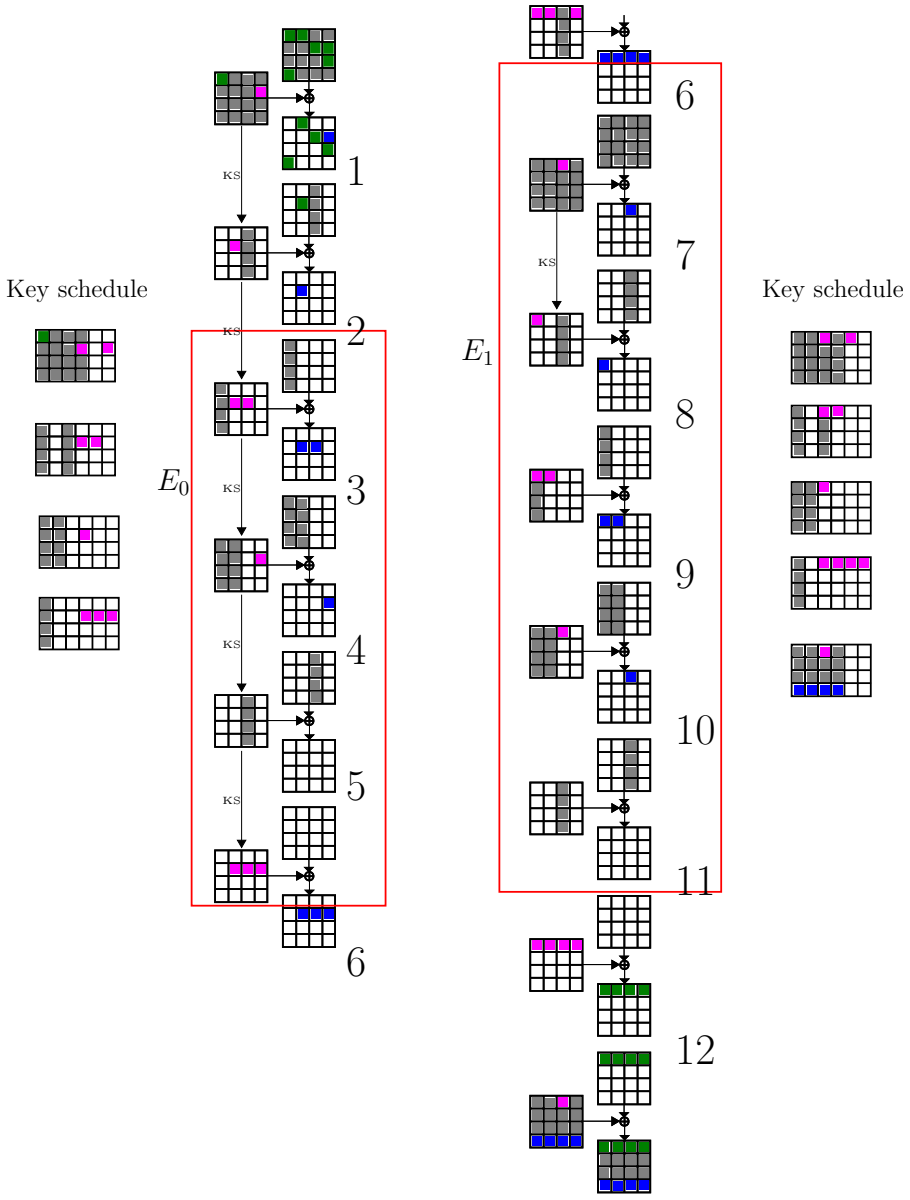

Figure 9.4: A related-key boomerang attack on AES-192. The bottom characteristic is the same as in [28].

differential on 11 rounds [91]. The key schedule of Camellia is not byte oriented because the rotation constants are not a multiple of 8. In order to test our tool we will make it byte oriented, by using the following rotation constants for rounds 1-18: 0, 0, 16, 16, 16, 16, 48, 48, 48, 64, 64, 64, 96, 96, 96, 96, 112, 112. We call this version — byte-Camellia. Note that, since we choose the rotation constants as close as possible to the original constants, a differential characteristic in the key schedule of byte-Camellia, may be suitable for the original Camellia. For that to happen, the positions of the active *bits* in an active byte have to be invariant of small rotations. On the other hand, trying all possible combinations of active bits, i.e. building all possible differential characteristics in the key schedule for the original version of Camellia, seems too much time consuming. Hence, we will analyze only byte-Camellia.

The key schedule of Camellia-128 applies transforms (4 rounds) to the master key K_L, to produce another key K_A and it uses these two values to generate the subkeys in a linear way. Therefore, *we will use variant 3 of the tool*. Internally in the tool, in all steps we will use only the S-type representation.

9.3.1 Best Characteristics for byte-Camellia

The maximal difference propagation probability of the S-boxes in Camellia is 2^{-6}. Therefore, we can allow no more than $\lfloor \frac{128}{6} \rfloor = 21$ active S-box in the characteristic of the key and the state. With this type of limitations, the tool produced the best related-key differential characteristic. It is on 8 rounds, and it has 20 active S-boxes.

9.3.2 Chosen-key Attack on Full-round byte-Camellia

When searching for chosen related-key characteristics in byte-Camellia, we can spend 21 active S-box in each, the key and the state (using the divide-and-conquer technique). With these weight limitations our tool was able to produce a good characteristic on all 18 rounds of byte-Camellia. The characteristic has 17 active S-boxes in the key, and 15 in the state (see Fig. 9.5). The characteristic can be used to show that Davies-Meyer hash function construction initiated with byte-Camellia-128 cipher, can be distinguished from a random function.

9.4 Khazad

Khazad [13] is a 64-bit block cipher with a key size of 128 bits. It is an SPN with 8 rounds. The best attacks go only up to 5 rounds: an integral attack [103] with 2^{91} complexity and a class of 2^{64} weak keys which can be attacked in 2^{40} steps using a slide attack [24].

The subkeys in the key schedule of Khazad are obtained consecutively from one another using a Feistel function. Therefore, *we will use variant 2 of the tool*. The small key and block sizes, in addition to the low branching in the key schedule allows to use the variant 1 as well. The optimal representation is similar to the one used in the tool for AES. In the state, after the S-boxes (γ) we will have non-zero only S-value, and after the linear-diffusion layer (θ) only P-value.

9.4.1 Best Characteristics for Khazad

The maximal difference propagation of the S-boxes in Khazad is 2^{-5}. Hence, a differential characteristic for Khazad cannot have more than 25 active S-boxes, at most 12 can be in

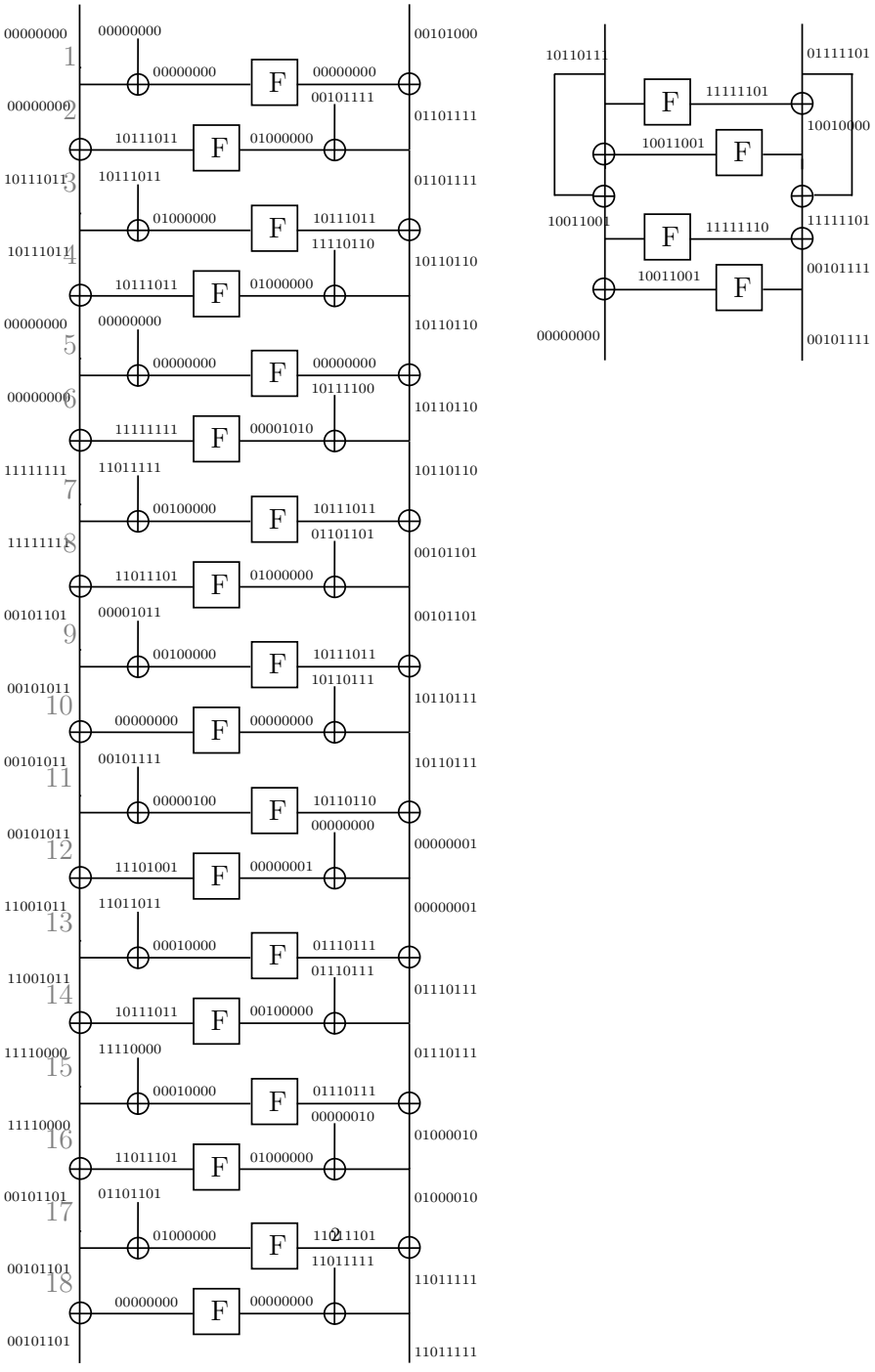

Figure 9.5: The related-key differential characteristic (left in the state, right in the key) on full-round byte-Camellia-128 for the chosen-key distinguisher. The characteristic has compact representation, the actual differences are to be fixed. The key XOR is depicted separately from the function F.

the state. With this type of limitations, the tool was able to produce interesting results. The best related-key differential characteristics for 4,5,6, and 7 rounds have 9, 10, 19, and 20 active S-boxes, respectively. The related-key attacks based on such characteristics would be the new best attacks on Khazad up to 7 rounds. The 7-round characteristic is presented at Fig.9.6.

9.4.2 Related-key Boomerang Attacks on 7 rounds of Khazad

Let us improve the probability of the 7-round attack by using a boomerang attack. We will use two 4 round characteristics (See Fig.9.7). The four related keys K^A, K^B, K^C, and K^D, are obtained as follows: 1) fix any K^A, i.e. (K^A_{-2}, K^A_{-1}), 2) produce (K^A_0, K^A_1) from K^A, and fix K^B such that $K^B = (K^A_0, K^A_1) \oplus (\Delta K_0, \Delta K_1)$, 2) obtain (K^A_6, K^A_7) and (K^B_6, K^B_7) and then fix $K^C = (K^A_6, K^A_7) \oplus (\Delta K_6, \Delta K_7)$, $K^D = (K^B_6, K^B_7) \oplus (\Delta K_6, \Delta K_7)$. The pink difference was chosen such that after γ and θ it could produce gray difference with a probability 2^{-5}. Let us find the complexity of the attack. We start with the same one byte difference (the pink byte) in the plaintext and the subkey K_0, hence there are no active bytes in the state in round 1 and 2. The difference in the subkey K_3, as well as in the state, (denoted with the grey bytes) is obtained when the pink byte goes through γ and θ, and hence it happens with 2^{-5}. At the end of round 4 we can switch to the bottom characteristic. The ciphertext difference is fully determined. We pay 2^{-5} in round 7 so the blue byte in the state after the inverse S-box will become pink (and then cancel with the pink difference in the key). To get a zero difference in the subkey K_5 we pay additional 2^{-5}. In round 4 we switch the state to the top characteristic. An important moment is the switch in the keys. When the gray difference in the top characteristic between K^A and K^B in the subkey K_3 is the same as the gray difference in the bottom characteristic between K^A and K^C (and K^B and K^D) in the subkey K_3, then the switch in the key is for free (this is due to the Feistel switch[5], See [28]). Then not only the difference in K_3 between K^C and K^D will be the same as between K^A and K^B, but their value will be equal to the values of K^A_3 and K^B_3 and hence will go through the S-boxes producing the same values. Therefore, we pay additional 2^{-5} for each of the differences in subkey K_3 (instead of a switching cost of 2^{-64}!). After the switch to the top characteristic, we pay 2^{-5} in the state of round 3 to get the same pink difference which will cancel after the key XOR. We pay additional 2^{-5} for the zero difference in the subkey K_1. The rest of the characteristic holds with probability 1. The probability of the whole boomerang attack is $2^{-(2\cdot5+2\cdot5+2\cdot5+2\cdot5+5+5)} = 2^{-50}$. This translates into a boomerang attack in a class of weak keys that works for 1 out of 2^{30} related-key quartets, with a complexity 2^{20} encryptions/decryptions. Moreover if we relax constraints on the difference in the key we can increase the size of the weak key class to 1 out of every 2^8 keys which can be attacked with complexity of 2^{49} encryptions and analysis steps. In both cases when the boomerang returns we know the plaintext difference and thus we have a 64 bit filter which allows us to filter out all the wrong quartets. Returning boomerang provides us with 7 bits of information about the key byte K^0_2 since we know the input and output difference for the active S-box of the key schedule and similarly about 7 bits of the key of K^0_6. One can extend this attack into a full key recovery attack via auxiliary techniques.

[5] We have tested the Feistel switch in the key schedule of Khazad, and a related-key quartet, following the whole 7-round differential characteristic, was found.

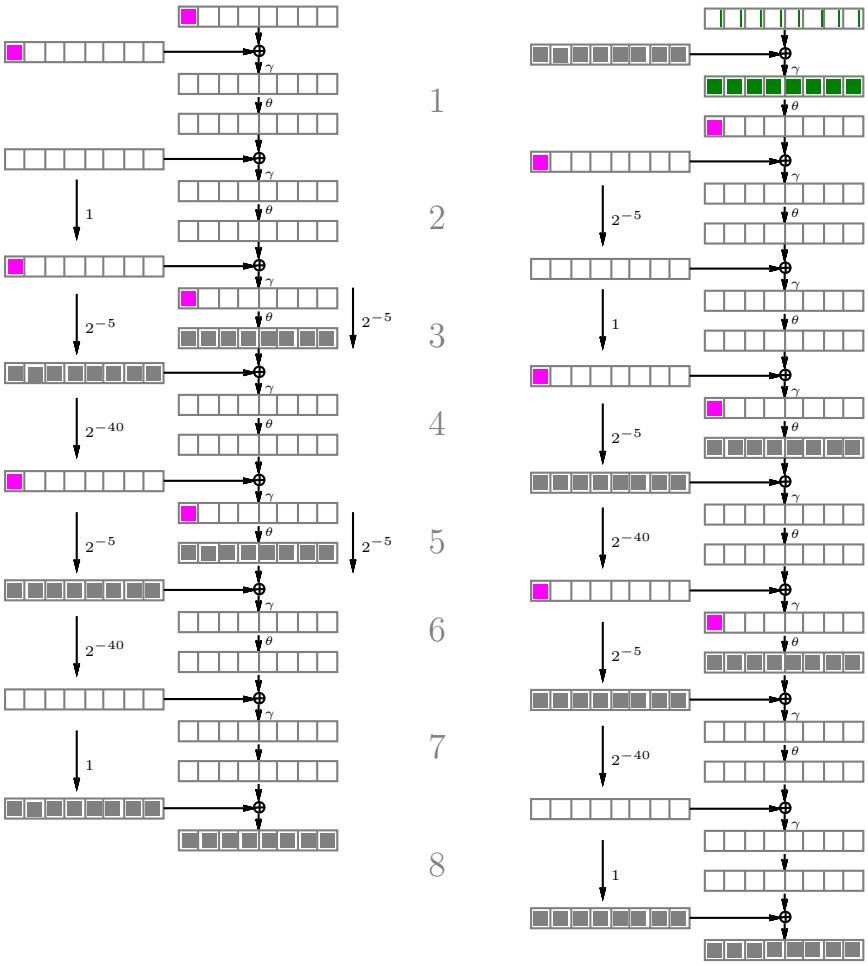

Figure 9.6: Related-key differential characteristic on 7 rounds (left), and related-key differential characteristic on 8 rounds (right) used for a chosen-key distinguisher.

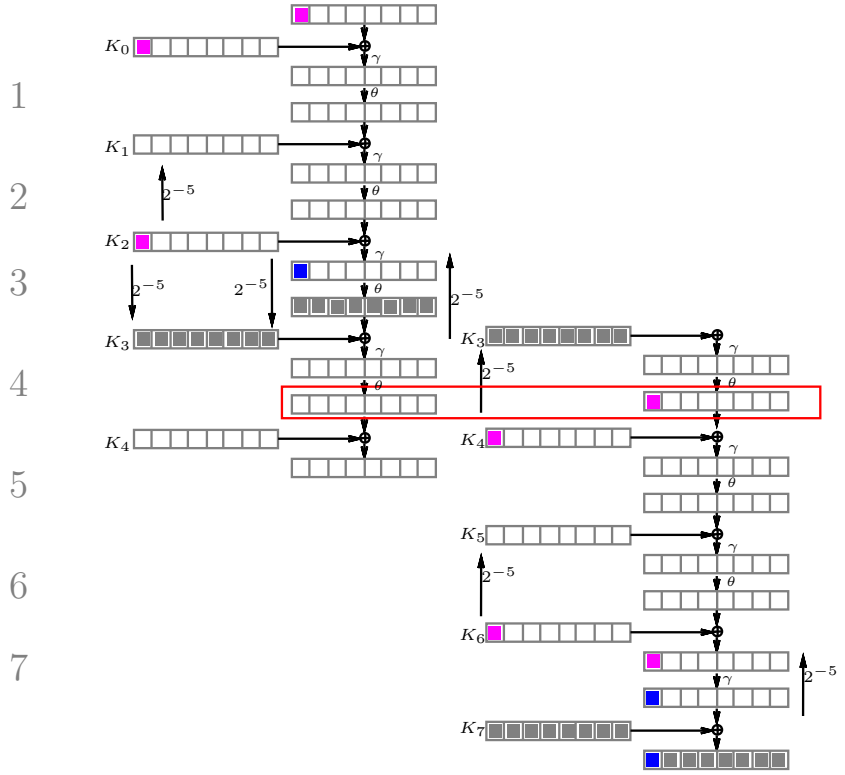

Figure 9.7: Characteristics for the RK boomerang attack on 7-round Khazad

9.4.3 Chosen-key Attack on Full-round Khazad

The 7-round related-key differential characteristic can easily be extended at the top for an additional round and then used in a chosen-key attack (See Fig. 9.6). Since we control the exact values of the key and the state, we will use the divide-and-conquer technique, and first fix the keys satisfying the characteristic in the key, and then find a proper pair of plaintexts that follow the characteristic in the state. We can use the rebound attack [98] and fix one round for free in both the key and in the state. For the key, we can fix the round for ΔK_4 (or ΔK_6), and obtain a characteristic in the key that holds with probability 2^{-55}. In the state, we will fix the values in the first round, hence the characteristic in the state holds with 2^{-10}. The S-boxes are non-injective regarding the difference, i.e. if we fix the input and output difference of the S-box, then there is a solution with probability $\frac{1}{2}$. Therefore, we introduce a possible one bit difference in each byte of the plaintext so that there will always be a solution for the S-box input/output differences. The total complexity of the chosen-key distinguisher is bounded by the probability of the characteristic in the key and is 2^{55}. The input difference is fixed in 56 bits, while the output is fixed in all 64-bits. This shows that full Khazad has properties which are not present in an ideal cipher. This also means, for example, that 256-bit Tandem-DM[86] hash function construction initiated with Khazad cipher, can be distinguished from a random function.

9.5 xAES

In the latest cryptanalytical results on AES [29, 28], it was shown that AES-192 and AES-256, i.e. the versions of AES with 192 and 256 key bits, do not have the ideal security level in the framework where related-key attacks are permitted. Despite the fact that so far these attacks are only theoretical and require a computational power beyond our reach, finding an efficient fix for AES that will produce a cipher that is ideal by the cryptographic standards, seems a good open problem. Further, we propose such fix in a form of a new cipher which we call xAES (x stands for extended security). Since the recent attacks are mostly based on the property of the key schedule of AES, we tweak only this part of the cipher, while keeping intact the round function. We introduce only a few additional operations in the key schedule which result in a cipher that is: 1) resistant against related-key differential attacks, 2) has a speed close to the speed of AES.

9.5.1 Specification of xAES

Let us give a complete specification of xAES. Similarly to AES, xAES supports three key sizes: 128, 192, 256, denoted as xAES-128, xAES-192, and xAES-256 respectively. Although in the section on AES we have found that no differential characteristic exist on the full round AES-128, we introduce xAES-128 to have a complete family of ciphers supporting the standard key sizes of 128,192, and 256 bits. The number of internal rounds in xAES for different key sizes is the same as the number of rounds in AES, i.e. 10 rounds for xAES-128, 12 rounds for xAES-192, and 14 rounds for xAES-256. Each internal round is defined same as in AES – through the four transformations SubBytes, ShiftRows, MixColumns and AddRoundKey. The only difference between AES and xAES is in the key schedule. Yet, the difference is small. In short, for obtaining each next column of the new subkey, xAES *always* uses rotation by one byte up of the previous subkey column, while AES uses a rotation only when obtaining the subkey column with an index multiple of N_k ($N_k = 4, 6, 8$ for AES-128,-192,-256). Let us give a formal definition of the new key schedules. We assume that the master key K is given

as an array $K[4][N_k]$ and the key schedule produces a subkey array $W[4][4(N_r+1)]$. The s-th subkey is given by the columns $4\cdot s$ to $4\cdot(s+1)-1$ of W. The round constant $RC[i][j]$ is the same as in AES.

The key schedule of xAES-128 is defined as follows. Let $K[4][4]$ be the master key. Then the subkey array $W[4][44]$ is defined as:

$$W[i][j] = \begin{cases} K[i][j], & \text{if } j < 4 \\ S(W[i-1 \bmod 4][j-1]) \oplus W[i][j-4] \oplus RC[i][j/4], & \text{if } j \bmod 4 == 0 \\ W[i-1 \bmod 4][j-1] \oplus W[i][j-4], & \text{otherwise} \end{cases}$$

The key schedule of xAES-192 is defined as follows. Let $K[4][6]$ be the master key. Then the subkey array $W[4][52]$ is defined as:

$$W[i][j] = \begin{cases} K[i][j], & \text{if } j < 6 \\ S(W[i-1 \bmod 4][j-1]) \oplus W[i][j-6] \oplus RC[i][j/4], & \text{if } j \bmod 6 == 0 \\ S(W[i-1 \bmod 4][j-1]) \oplus W[i][j-6], & \text{if } j \bmod 6 == 3 \\ W[i-1 \bmod 4][j-1] \oplus W[i][j-6], & \text{otherwise} \end{cases}$$

The key schedule of xAES-256 is defined as follows. Let $K[4][8]$ be the master key. Then the subkey array $W[4][60]$ is defined as:

$$W[i][j] = \begin{cases} K[i][j], & \text{if } j < 8 \\ S(W[i-1 \bmod 4][j-1]) \oplus W[i][j-8] \oplus RC[i][j/4], & \text{if } j \bmod 8 == 0 \\ S(W[i-1 \bmod 4][j-1]) \oplus W[i][j-8], & \text{if } j \bmod 8 == 4 \\ W[i-1 \bmod 4][j-1] \oplus W[i][j-8], & \text{otherwise} \end{cases}$$

To clearly understand the idea of the additional operations, in Fig. 9.8 we give a pictorial representation of how the $s+1$-th subkey is obtained from the s-th subkey, for all three key schedules of xAES as well as for the key schedule of AES-256 so the reader can compare the changes.

Besides the additional rotations, the difference between AES and xAES is in the 192-bit key version. We introduce an additional layer of S-boxes to meet our security level.

9.5.2 Efficiency of xAES

Let us compare the speed of xAES with the speed of AES in software. We assume that both use optimal implementation with table lookups and xors. First let us give a theoretical comparison of the efficiency. Recall that since the round functions of AES and xAES are identical, in an environment where the master key is fixed and the encrypted message is longer (encryption mode), their speed is the same. Now let assume that the master key is frequently changed (hash mode). One round of AES (and therefore of xAES) has 16 table lookups and 16 xors.

One out of 10 subkey rounds of AES-128 has 4 table lookups and 8 xors. In xAES one such round has 4 table lookups, 8 xors and 3 rotations. If we assume that rotations have the same cost as xors, then for encrypting 16 bytes, AES uses $10\cdot 16+10\cdot 4=200$ lookups and $10\cdot 16+10\cdot 8=240$ operations. In xAES these numbers are $10\cdot 16+10\cdot 4=200$ lookups and $10\cdot 16+10\cdot 4+10\cdot 3=270$ operations.

In AES-192, one subkey round (out of 8) has 4 table lookups and 10 xors while in xAES-192 one subkey round has 8 table lookups, 10 xors and 5 rotations. Hence, for 16 bytes,

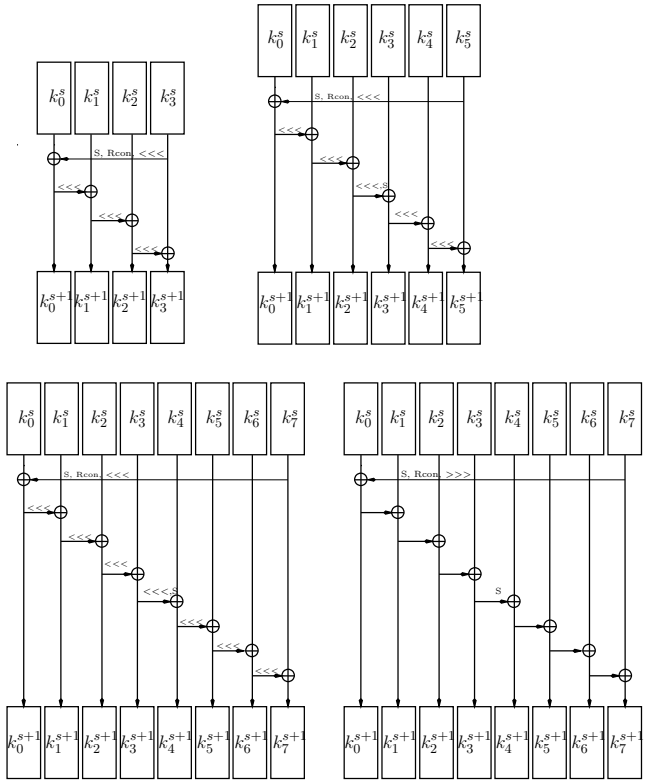

Figure 9.8: One subkey round for xAES-128 (top left), xAES-192 (top right), xAES-256 (bottom-left), and AES-256 (bottom-right). Each k_i^j is a subkey column – 4 bytes. $\lll$ ($\ggg$) stand for a word rotation on 8 bits to the left(right), S for S-box, and Rcon for xor of the round constant.

AES-192 spends $12 \cdot 16 + 8 \cdot 4 = 224$ lookups and $12 \cdot 16 + 8 \cdot 10 = 272$ operations, while xAES-192 spends $12 \cdot 16 + 8 \cdot 8 = 256$ lookups and 312 operations.

Finally, AES-256 has 7 subkey rounds and in each it has 8 table lookups and 16 xors. xAES-256 has the same number of subkey rounds and in each uses 8 table lookups , 16 xors and 7 rotations. For 16 bytes, AES-256 uses $14 \cdot 16 + 7 \cdot 8 = 280$ lookups and $14 \cdot 16 + 7 \cdot 16 = 336$ operations, while xAES-256 uses $14 \cdot 16 + 7 \cdot 8 = 280$ lookups and $14 \cdot 16 + 7 \cdot 16 + 7 \cdot 7 = 385$ operations.

These numbers indicate that one can expect that xAES-128 is around 6%, xAES-192 is around 12.5%, and xAES-256 is 7% slower than AES-128, AES-192, and AES-256 respectively. The actual implementation results obtained based on optimal implementation on C, together with the theoretical estimates, are given in Tbl. 9.3. In the table, the "Hash mode" entries indicate the speed of xAES compared to the speed of AES where the master key is changed on every iteration (16 bytes of plaintext), while the "Encryption mode" entries

Table 9.3: Comparison of the speed of xAES with the speed of AES. The abbreviations "tb" and "op" stand for table lookups and operations, respectively.

xAES/AES			
	128	192	256
	Hash mode		
Theoretical	$\frac{200\text{tl}+240\text{op}}{200\text{tl}+270\text{op}} \approx \approx 94\%$	$\frac{224\text{tl}+272\text{op}}{256\text{tl}+312\text{op}} \approx \approx 87\%$	$\frac{280\text{tl}+336\text{op}}{280\text{tl}+385\text{op}} \approx \approx 93\%$
Implementation	96%	83%	97%
	Encryption mode		
Theoretical, Implementation	100%	100%	100%

compare the speed when the master key is fixed[6]. The difference between the theoretical estimate and the actual implementation comes from the fact that the modern processors in one clock cycle can perform several xor operations but only one table lookup.

9.5.3 Related-key Differential Attacks in xAES

Since the only difference between AES and xAES is in the key schedule, to implement the tool for xAES, we have to find method that checks $\Delta K_{new} == SubkeyRound(\Delta K_{old})$, i.e. if the difference in the next subkey can be obtained from the difference of the previous subkey. Taking into consideration that the subkeys columns are produced one by one (see Fig. 9.8), this problem is reduced to the problem of checking if the differences in three subkey columns $k^i_j, k^i_{j+1}, k^{i-1}_{j+1}$ (where k^i_j, k^i_{j+1} are the j and $j+1$-th columns of the i-th subkey, and k^{i-1}_{j+1} is the $j+1$-th column of the $i-1$-th subkey) are related as:

$$(\Delta k^i_j) \lll 8 \oplus \Delta k^i_{j+1} = \Delta k^{i-1}_{j+1}. \tag{9.1}$$

Let us focus on the compact representation of a difference in the subkey column. This difference is described with an 8-bit vector $\tilde{c} = (\tilde{a}, \tilde{b}) = (a_1, \ldots, a_4, b_1, \ldots, b_4), a_i, b_i \in \{0,1\}$. The compact $\tilde{c}$ describes (is a set of) all subkey column differences $\Delta\tilde{d} = (\Delta d_1, \Delta d_1, \Delta d_2, \Delta d_3)$, $\Delta d_i \in Z_{2^8}$, such that:

$$\Delta\tilde{d}^T = MC \cdot (x_1, x_2, x_3, x_4)^T \oplus (y_1, y_2, y_3, y_4)^T = MC \cdot \tilde{x}^T \oplus \tilde{y}^T, \tag{9.2}$$

where MC is the matrix of the MixColumns transform, $x_i, y_i \in Z_{2^8}$ and $x_i > 0$ **iff** $a_i > 0$, $y_i > 0$ **iff** $b_i > 0$. Note that for example the vector $(\tilde{0}, \tilde{b})$ is a simple truncated representation of a difference. In our implementation we will use the same compact representation for the differences in the subkey columns. To check (9.1) we have to deal with the rotation of the key Δk^i_j since in the key schedule of AES there is no such rotation. Let us see how this

[6]For short messages in the encryption mode, refer to the speed of "Hash mode".

rotation influences the compact representation. If the difference Δk_j^i in the subkey is $\Delta\tilde{d}$ (see (9.2)), then the rotation of this difference is:

$$\Delta\tilde{d}^T \lll 8 = (MC \cdot \tilde{x}^T \oplus \tilde{y}^T) \lll 8 = (MC \cdot \tilde{x}^T) \lll 8 \oplus \tilde{y}^T \lll 8 \tag{9.3}$$

The matrix MC has the property $(MC \cdot X) \lll 8 = MC \cdot (X \lll 8)$. Hence (9.3) can be rewritten as:

$$\Delta\tilde{d}^T \lll 8 = MC \cdot (\tilde{x}^T \lll 8) \oplus \tilde{y}^T \lll 8 \tag{9.4}$$

Therefore, if the initial difference Δk_j^i had a compact representation $\tilde{c_1} = (a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4)$ then the rotation of this difference $\Delta k_j^i \lll 8$ has a compact representation $\tilde{c_1} = (a_2, a_3, a_4, a_1, b_2, b_3, b_4, b_1)$. Hence, all the differences in (9.1) have a compact representation and this condition can easily be checked[7].

We have implemented the tool in practice and we searched for the best *round-reduced* related-key differential characteristics. To prove the resistance of r-round xAES against related-key differential attacks, these characteristics have to have certain properties:

1. No two characteristics exist with probability $2^{-p_1}, 2^{-p_2}$ on r_1 and r_2 rounds such that $r_1 + r_2 \geq r - 2$ and $2p_1 + 2p_2 \leq k$, where k is the key size. This restriction was introduced to stop the boomerang attacks on the full r rounds. We assume that two rounds can be obtained for free by various techniques, but the rest $r_1 + r_2$ rounds are part of the boomerang.
2. No characteristics exist on $r/2$ rounds with probability higher than $2^{-k/2}$. Obviously, this was introduced to stop the related-key differential attacks on the full r-round cipher which always can be seen as a concatenation of two $r/2$-round ciphers.

Each characteristic found by the tool is presented in a compact form. To find the probability of a characteristic one has to count the number of active bytes, i.e. the position of the bytes with 1 that go through the S-boxes. For example, a characteristics with s active bytes has a probability at most $2^{-6 \cdot s}$ because the maximal differential propagation of an S-box in AES is 2^{-6}. In table Tbl. 9.4 we give the probabilities (in terms of active bytes) of the best related-key differential characteristics for xAES-128, xAES-192, and xAES-256[8]. Using these results we can prove the differential resistance of xAES.

In case of xAES-128, to have a valid differential characteristic[9], the number of active bytes in the differential attack should not exceed $\lfloor \frac{128}{6} \rfloor = 21$ (because the key size is 128 bits and the maximal differential propagation of the S-box is 2^{-6}). In a boomerang attack (recall that since xAES-128 has 10 rounds, we try to build a boomerang on $10 - 2 = 8$ rounds), at least one of characteristics (upper or lower) has to be on 4 rounds, for which the attacker has to pay at least $2 \cdot 10 = 20$ active S-boxes. Hence, he has only $21 - 20 = 1$ active S-box left which is insufficient for a boomerang on 8 rounds and therefore xAES-128 is resistant against boomerang-type attacks. Now let us try to build a differential characteristic on all 10 rounds. Note that the characteristic on 5 rounds has more than 11 active S-boxes, hence the characteristic on 10 rounds would have more than $2 \cdot 11 = 22$, and therefore no related-key differential characteristic exist on all 10 rounds of xAES-128.

In xAES-192, the number of active S-boxes in a valid differential characteristic is bounded by $\lfloor \frac{192}{6} \rfloor = 32$. For a boomerang attack, in case one of the characteristics has 6 or more

[7] Note that now checking (9.1) is reduced to checking for the case where all three subkey columns have a compact representation, which was solved for AES.

[8] For example, the best differential characteristic for xAES-192 on 5 rounds has 9 active S-boxes – in Tbl. 9.4 the intersection of row 5 and column xAES-192 is 9.

[9] A valid characteristic has a probability higher than 2^{-128}.

Table 9.4: The number of active S-boxes in the best round-reduced related-key differential characteristics in xAES-128, xAES-192, and xAES-256.

$rounds$	$xAES-128$	$xAES-192$	$xAES-256$
2	1	0	0
3	5	1	1
4	10	4	3
5	>11	9	7
6	>11	>16	13
7	>11	>16	18
8	>11	>16	>21

rounds, the number of active S-boxes becomes greater than 32, hence the attack does not work. When both characteristics have 5 rounds, then this number is $2\cdot 9+2\cdot 9=36$ which is again higher than 32, and therefore the boomerang with 10 rounds has lower probability than 2^{-192}. Similarly, any differential characteristic on 12 rounds (which can be seen as $6+6$ rounds), has more than $16+16=32$ active S-boxes, hence its probability is less than 2^{-192}.

In xAES-256, the number of active S-boxes is bounded by $\lfloor\frac{256}{6}\rfloor=42$. Regarding the boomerang attack, when one of the characteristics is on at least 8 rounds, the attacker only for this characteristic has to pay more then $2\cdot 21=42$ active S-boxes. When the characteristics are on 7 and 5 rounds, then the attacker pays $2\cdot 18+2\cdot 7=50$, while when both are on 6 rounds, he pays $2\cdot 13+2\cdot 13=52$ active S-boxes. In each of these cases, the total probability of the boomerang is less than 2^{-256}. The best characteristic on 7 rounds has only 18 active S-boxes. Hence, we cannot trivially prove that it does not exist a characteristic on 14 rounds (because $18+18=36$ which is less than 42). That is why we have to take a different approach. Any characteristic on 14 rounds, is composed of two 7-round characteristics, one of which has to have no more than 21 S-boxes (otherwise the total sum will be more than 42). First, we build all characteristics on 7 rounds that have no more than 21 active S-box. Then, we try to extend upward and downward each such characteristic for 7 additional rounds. Our search did not find any good candidate, i.e. no characteristic on 7 rounds with at most 21 active S-box can be extended to a characteristic on 14 rounds (with no more that 42 active S-boxes) and therefore no related-key differential characteristic exists on 14 rounds in xAES-256. This concludes our proof for the related-key differential resistance of full-round xAES.

The search for the best related-key differential characteristics for different version of xAES, i.e. running the tool in practice and obtaining the results of Tbl. 9.4, required different amount of computational effort. While finding the probabilities and the actual values for the round-reduced characteristics in xAES-128 and xAES-192 was performed in a few hours, in xAES-256 the search required a few days on a single core. The search for all good characteristics on 7 rounds of xAES-256 with no more than 21 active S-boxes produced around 2^{15} candidates and required two weeks on 8 cores. Extending these 7-round characteristics to 14 rounds, as mentioned before, did not give any good candidate, and required a few days on a single core.

Chapter 10

The Search in DES-like ciphers

The Data Encryption Standard (DES) [104], adopted by the U.S. National Bureau of Standards in 1977, was a block cipher standard for several decades. Some of the design principles of DES were fully understood by the public only after the first cryptanalysis presented by Biham and Shamir [22]. They introduced the idea of differential analysis and differential characteristics, and showed that if one encrypts with DES a pair of plaintexts with a specific XOR difference, then the pair of corresponding ciphertexts will have some predictable difference with a probability higher than expected.

In [93] Matsui showed that the differential characteristics found by Biham and Shamir were indeed the best, i.e. they have the highest probability among all characteristics. He was able to prove this fact by running a full search on the space of all possible characteristics, using a special algorithm that speeds up the search. Matsui's algorithm was adopted and applied for search of the best characteristics in LOKI and s^2DES [135], Twofish [131], FEAL [3], and others. In all these cases, the search was targeting only single-key characteristics, i.e. the characteristics that have a difference in the plaintext, but not in the master key.

We present algorithms for finding the best (with the highest probability) round-reduced related-key differential characteristics in DES and DES-like ciphers. We show that instead of trying all differences in the key and in the plaintext, which would result in a search space of size 2^{120}, it is computationally more efficient to try only a reduced set of input-output differences of three consecutive S-boxes layers. Based on this observation, we are able to propose two algorithms for automatic search of related-key differential characteristics in DES-like ciphers – the first is based on Matsui's approach, while the second is in line with the technique of divide-and-conquer. We apply our algorithms to DES, DESL [88], and s^2DES [82] and find either the probabilities of the best round-reduced related-key differential characteristics, or the upper bounds on these probabilities. Interestingly, although for lower number of rounds these probabilities are much higher than in the case of single-key characteristics, for higher number of rounds, the best characteristics are single-key characteristics. We obtain an interesting result regarding DES. By providing the probability of the best related-key characteristic on 13 rounds, we show that Biham-Shamir attack cannot be improved if one uses related-key characteristic (instead of single-key). Moreover, the low probabilities of the best related-key characteristics on higher rounds indicate that NSA did not introduce any weakness (or trapdoor) in the key schedule of DES with regard to differential attacks. Although in this paper we apply our algorithms only to the DES-like ciphers, we believe that

our approaches can be used as well to search for high probability related-key differential characteristics in any bit-oriented ciphers with linear key schedule.

10.1 Description of DES-like Block Ciphers

DES [104] is 64-bit block cipher with 56-bit key[1]. It is 16-round Feistel cipher with additional permutations IP, IP^{-1} at the beginning and at the end. The 64-bit plaintext, after the application of the initial permutation IP is divided into two halves L_0 and R_0 - each half has 32 bits. Then, the halves are updated 16 times with the round function:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i),$$

where $i = 1, \ldots, 16$ and K_i are 48-bit round keys, obtained from the initial key K with some linear transforms (rotations that depend on the round number and bit selection function PC-2). The ciphertext is defined as $IP^{-1}(R_{16}||L_{16})$.

The round function $f(R, K_i)$ takes 32-bit state R and 48-bit round key K_i and produces 32-bit output. First it expands the 32-bit value of R to 48 bits with the linear function E and then it XORs the values of $E(R)$ and K_i to produce some intermediate result, which we further denote as f_i. This 48-bit value is divided into 8 six-bit values, and each of these values goes through a separate 6x4 S-box. Finally, the 32-bit output of the S-boxes goes through a bit permutation P and the output $\tilde{f}_i$ of the round function is produced.

The DES-like block ciphers DESL [88] and s^2DES [82], differ from DES only in the definition of the S-boxes and the initial and final permutations. Since these permutations have no cryptographic values, we can assume that the only difference among the ciphers of the DES-family is in the S-boxes.

10.2 Automatic Search for Related-key Differential Characteristics in DES-like ciphers

The best characteristic on r rounds, i.e. the best r round-reduced characteristic, is the one that has the higher probability among all characteristics on r rounds of the cipher. In this section we propose two methods for building efficient automatic search algorithms for finding the best round-reduced related-key differential characteristics in DES-like ciphers. When constructing these algorithms, the main problem that has to be tackled is how to deal with the enormous search space. There are 64 bits in the state and 56 bits in the key, hence in total there are 2^{120} starting values for differential characteristics. However, in general, this number can be reduced significantly. Our first method is based on Matsui's search tool applied for finding the best single-key round-reduced characteristics in DES. The second method, which we call *the split* approach, can be used when Matsui's approach fails – to find characteristics on high number of rounds when not all the characteristics on lower number of rounds are known.

Due to the complementation property of DES, there are related-key characteristics (including round-reduced) that hold with probability 1. Further, we do not consider these characteristics.

[1] Officially, the key has 64 bits, but 8 bits are only used to check the parity, and then discarded.

Considering the different rotation amounts in the key-schedule, the probability of the best round-reduced related-key characteristic depends on the rounds covered by the characteristic. For example, the best 5-round related-key characteristic covering rounds 0-4, can have different probability from the best characteristic that covers rounds 1-5. The best related-key characteristics in our paper, always cover the last rounds, e.g. the characteristic on 7 rounds, covers the rounds 9-15.

10.2.1 Matsui's Approach for Single-key Characteristics

The search for the best single-key differential characteristics in DES was successfully performed by Matsui in [93]. Note that even in this case, when there is no difference in the key, the search space is rather large – 2^{64} starting differences. However, Matsui presented several useful approaches how to deal with a large number of starting differences and how to significantly reduce the search space.

A naive approach to search for the best n-round characteristic would be to try all possible starting differences in the plaintext and try to extend each of them to n rounds. The non-linearity of the S-boxes will introduce branching, and a k-round characteristic ($k < n$) is extended for an additional round only if its probability is higher than the probability P_n^* of some known characteristic on n rounds.

Matsui's approach on the other hand, cuts out a large number of round-reduced characteristics in the early stage. Given the probabilities $\overline{P_1}, \ldots, \overline{P_{n-1}}$ of the best characteristics on the first $n-1$ rounds, and some estimate[2] P_n^* for the probability of the characteristic on n rounds, the algorithm produces the best characteristic on n rounds. Hence, the attacker can sequentially produce, starting from 1 or 2-round reduced, characteristics on all rounds of DES. In short, the attacker, as in the naive approach, tries all possible starting differences[3]. For each of them he produces 1-round characteristic (there can be many one-round characteristics, and the following procedure is repeated for each of them) that holds with probability P_1. Then, he tries to extend it to two rounds only if $P_1 \cdot \overline{P_{n-1}} > P_n^*$. This is because in order to extend 1-round characteristic to n rounds, one should use an additional $(n-1)$-round characteristic. Since the best one has probability $\overline{P_{n-1}}$, the total probability of the n-round characteristic will be at most $P_1 \cdot \overline{P_{n-1}}$ and this value should be better than the probability P_n^* of the best known characteristic on n-rounds. Similarly, if the attacker has built k-round characteristic with probability P_k than he tries to extended for an additional round only if $P_k \cdot \overline{P_{n-k}} > P_n^*$. Note that in the naive approach, the attacker only checks if $P_k > P_n^*$. Therefore, Matsui's approach stops the extension of many round-reduced characteristics and that way speeds up the search.

Now let us take a closer look how to reduce the number of possible starting differences. Interestingly, the same approach as above can be used. First note that a characteristic on the first two rounds (assuming this 2-round characteristic is part of the best n-round characteristic) has a probability P_2 such that $P_2 < P_n^*/\overline{P_{n-2}}$. The following observation is used to explore this property of 2-round single-key characteristics.

Observation 1 *Given the input and the output differences* $(\Delta f_1, \Delta \tilde{f_1}), (\Delta f_2, \Delta \tilde{f_2})$ *of the S-boxes layers in the first two rounds, one can find the difference in the plaintext* ΔP *and the difference* $(\Delta L_2, \Delta R_2)$ *in state at the beginning of the third round.*

Proof. From the Feistel construction it leads that $\Delta R_0 = E^{-1}(\Delta f_1)$ and $\Delta L_0 = \Delta \tilde{f_1} \oplus E^{-1}(\Delta f_2)$. Then the difference ΔP in the plaintext is $\Delta P = IP^{-1}(\Delta R_0 || \Delta L_0)$. Similarly,

[2] For example, the attacker can use the probability of the already known characteristic on n rounds as an estimate.

[3] We will see later, that this requirement can be omitted.

for the difference at the beginning of the third round we get $\Delta R_2 = \Delta R_0 \oplus \Delta \tilde{f}_2$ and $\Delta L_2 = \Delta L_0 \oplus E^{-1}(\Delta f_2)$. □

Therefore, instead of fixing all possible differences ΔP in the plaintext one can fix only the input and the output differences to the S-boxes in rounds 1,2. But, since the active S-boxes of the first round have to hold with a probability of at least $P_n^*/\overline{P_{n-1}}$, and in the first and the second round with at least $P_n^*/\overline{P_{n-2}}$, the number of 2-round characteristics is significantly reduced. For each such characteristic, one can proceed with Matsui's technique, and try to extend it to n-rounds (since the difference at the beginning of round 3 is fixed).

10.2.2 Applying Matsui's Approach for Related-key Characteristics

One can easily reconstruct Matsui's algorithm to search for related-key characteristics. Note that for a fixed difference in the key, the algorithm still works and it finds the best characteristic with this specific difference. However, since the key has 56 bits, this search has to be repeated 2^{56} times and hence this naive approach is not feasible. We can still run a so-called *limited search* for related-key characteristics, by allowing low Hamming difference in the key. For example, to find the best characteristic that has at most 2-bit difference in the key, we have to rerun Matsui's algorithm $1 + C_{56}^1 + C_{56}^2 = 1597$ times.

Indeed, finding the best related-key characteristic using Matsui's approach can be done efficiently. We only have to find a way to efficiently limit the number of possible differences in the key and in the plaintext. We want to reduce the search space, yet to perform a full search of all possible related-key differential characteristics. The following observation can be used for that purpose.

Observation 2 *Given the input and the output differences* $(\Delta f_1, \Delta \tilde{f}_1)$, $(\Delta f_2, \Delta \tilde{f}_2)$, $(\Delta f_3, \Delta \tilde{f}_3)$ *of the S-boxes layers in the first three rounds, one can find the difference in the plaintext* ΔP, *the difference* $(\Delta L_3, \Delta R_3)$ *in state at the beginning of the fourth round, and all* 2^8 *values for the difference* ΔK *in the master key.*

Proof. Again we use the property of the Feistel construction and the linearity of the key schedule. From the definition of DES we get:

$$\Delta f_1 = E(\Delta R_0) \oplus \Delta K_1 \tag{10.1}$$

$$\Delta f_3 = E(\Delta R_0 \oplus \Delta \tilde{f}_2) \oplus \Delta K_3 \tag{10.2}$$

Since E is linear, we get:

$$\Delta K_1 \oplus \Delta K_3 = \Delta f_1 \oplus \Delta f_3 \oplus E(\Delta \tilde{f}_2)$$

The key schedule is linear, and both K_1 and K_3 are obtained from the master K with some linear transformation. Therefore $\Delta K_1 \oplus \Delta K_3$ can be expressed as $\mathscr{L}(\Delta K)$, where $\mathscr{L}$ is a linear transformation. On the other hand, the input-output differences of the S-boxes are given, and therefore, the value $V = \Delta f_1 \oplus \Delta f_3 \oplus E(\Delta \tilde{f}_2)$ is known. Hence, the master key difference ΔK can be found as $\Delta K = \mathscr{L}^{-1}(V)$. However, the key is 56 bits, while V only 48 bits. Therefore we get an underdefined system of linear equations with 2^8 solutions. If we fix a particular solution for the system, and thereby the difference in the key K, we can easily find $\Delta K_1, \Delta K_3$ (and ΔK_2). Then $\Delta R_0 = E^{-1}(\Delta f_1 \oplus \Delta K_1)$ and $\Delta L_0 = E^{-1}(\Delta f_2 \oplus K_2) \oplus \Delta \tilde{f}_1$. Similarly can be found the differences $\Delta L_3, \Delta R_3$. □

The above observation clearly indicates how to reduce the search space. Instead of trying all possible differences in the key K and running Matsui's algorithm for each of them, one should only fix the input and the output differences to the S-box layers in the first three rounds. Due to restrictions on the probability, all the active S-boxes in first, in the first and second, and in the first, second and third round, should have a combined probability of at least $P_n^*/\overline{P_{n-1}}, P_n^*/\overline{P_{n-2}}, P_n^*/\overline{P_{n-3}}$, respectively. Once the active S-boxes for the first three rounds are fixed, one can easily find all 2^8 candidates for the difference in the master key and the difference in the state after the third round and hence produce 3-round differential characteristic with a fixed difference in the master key. Further, Matsui's approach can be used, and this characteristic can be extended to any number of rounds. The pseudo-code of the whole algorithm is given at Alg. 4.

10.2.2.1 On the complexity and optimization of the search.

Calculating the exact time complexity of the whole search is complex and probably impossible. However, some estimate can be given, under a certain assumption. Our experiments indicate that once the difference in the state (after the third round) and in the key is fixed, extending the characteristic to n rounds becomes fairly easy and computationally cheap task. The main complexity lies in generating all 3-round related-key characteristics that have a certain probability. More precisely, from observation 2 it follows that one should generate all active S-boxes in the first round that hold with a combined probability P_1 of not less than $P_n^*/\overline{P_{n-1}}$, then all active S-boxes in the second round with a combined probability not less than $P_n^*/(\overline{P_{n-2}} \cdot P_1)$ and all active in the third round with probability of not less than $P_n^*/(\overline{P_{n-3}} \cdot P_2)$ (where P_2 is the probability of the active S-boxes in the first two rounds). Therefore, the number of all 3-round related-key characteristics depends only on the values $P_n^*/\overline{P_{n-1}}, P_n^*/\overline{P_{n-2}}$ and $P_n^*/\overline{P_{n-3}}$ – higher the values, less characteristics exist, and the search is faster.

The complexity of creating all these 3-round characteristics is not the same (or proportional) as the number of such characteristics. This comes from the fact that the linear transform E is not a surjective, since it has 32-bit input and 48-bit output. For example, after ΔK_1 is found (see the proof of the observation 2), the value $\Delta R_0 = E^{-1}(\Delta f_1 \oplus \Delta K_1)$ exists only with a probability 2^{-16}. Similar holds for ΔL_0. Hence, the optimal strategy for creating the 3-round characteristics would be to:

1. Fix the probabilities of the first four active S-boxes in the first and the third round and all the active S-boxes of the second round (that have the above limitations), without fixing the exact input-output differences. This can be done by fixing only the possible values from the difference distribution tables of the S-boxes.
2. Fix the input differences to the four S-boxes of round 1,3, and the output differences of the S-boxes of round 2 (that correspond to the previously fixed distribution values).
3. Find 28 bits of ΔK, then find 28 bits of ΔK_1 and check if there exist preimage of 24 bits of $\Delta f_1 \oplus \Delta K_1$ for E. This can be done, since the left and the right 28-bit halves of the key are independent.
4. If exists, fix the probabilities of the last four active S-boxes in the first and the third round.
5. Fix the input differences to these 8 S-boxes.
6. Find the rest 28 bits of ΔK, then of ΔK_1 and check if there exist preimage of last 24 bits of $\Delta f_1 \oplus \Delta K_1$ for E.

Algorithm 4 Search for RK differential characteristic

FullSearch()
{
// The first three rounds
for all $\Delta f_1 \to \Delta\tilde{f}_1 | P(\Delta f_1 \to \Delta\tilde{f}_1)\overline{P_{n-1}} > P_n^*$ **do**
  for all $\Delta f_2 \to \Delta\tilde{f}_2 | P(\Delta f_1 \to \Delta\tilde{f}_1)P(\Delta f_2 \to \Delta\tilde{f}_2)\overline{P_{n-2}} > P_n^*$ **do**
    for all $\Delta f_3 \to \Delta\tilde{f}_3 | P(\Delta f_1 \to \Delta\tilde{f}_1)P(\Delta f_2 \to \Delta\tilde{f}_2)P(\Delta f_3 \to \Delta\tilde{f}_3)\overline{P_{n-3}} > P_n^*$ **do**
      $V = \Delta f_1 \oplus \Delta f_3 \oplus E(\Delta\tilde{f}_2)$
      for all $\Delta K | \mathscr{L}(\Delta K) = V$ **do**
        $\Delta K_1 = PC2(rot(\Delta K, 1))$
        $\Delta K_2 = PC2(rot(\Delta K, 2))$
        if $E^{-1}(\Delta K_1 \oplus \Delta f_1)$ and $E^{-1}(\Delta K_2 \oplus \Delta f_2)$ **then**
          $\Delta R_0 = E^{-1}(\Delta K_1 \oplus \Delta f_1)$
          $\Delta L_0 = E^{-1}(\Delta K_2 \oplus \Delta f_2) \oplus \Delta\tilde{f}_1$
          $\Delta R_3 = \Delta L_0 \oplus \Delta\tilde{f}_1 \Delta\tilde{f}_3$
          $\Delta L_3 = \Delta R_0 \oplus \Delta\tilde{f}_2$
          Call NextRound(ΔL_3, ΔR_3, ΔK, $P(\Delta f_1 \to \Delta\tilde{f}_1)P(\Delta f_2 \to \Delta\tilde{f}_2)P(\Delta f_3 \to \Delta\tilde{f}_3)$, 4)
        end if
      end for
    end for
  end for
end for
}

NextRound($\Delta L, \Delta R, \Delta K, p, round$)
{
$\Delta K_r = PC2(rot(\Delta K, round))$
$\Delta f = \Delta K_r \oplus E(\Delta R)$
for all $\Delta f \to \Delta\tilde{f} | P(\Delta f \to \Delta\tilde{f}) \cdot p \cdot \overline{P_{n-round}} > P_n^*$ **do**
  $\Delta L_{new} = \Delta R$
  $\Delta R_{new} = \Delta L \oplus \Delta\tilde{f}$
  if $round == n$ **then**
    if $P(\Delta f \to \Delta\tilde{f}) \cdot p > P_n^*$ **then**
      $P_n^* = P(\Delta f \to \Delta\tilde{f}) \cdot p$
    end if
  else
    Call NextRound($\Delta L_{new}, \Delta R_{new}, \Delta K, P(\Delta f \to \Delta\tilde{f}) \cdot p, round + 1$)
  end if
end for
}

7. If exists, find ΔK_2, fix the input difference to the S-boxes in the second round and check if there exist a preimage of $\Delta f_2 \oplus K_2$ for E.
8. If exists, fix the output differences for the S-boxes of round 3 (it is not necessary to fix the outputs of S-boxes of round 1).

Although we cannot give a precise estimate for the complexity of creating all 3-round characteristics, we can give such estimates for some particular fixed values of P_n^*,$\overline{P_{n-1}}$,$\overline{P_{n-2}}$, and $\overline{P_{n-2}}$. For example, when $P_n^*/\overline{P_{n-1}} = 2^{-3}, P_n^*/\overline{P_{n-2}} = 2^{-6}, P_n^*/\overline{P_{n-3}} = 2^{-9}$, then steps 1-8 are repeated $2^{16.7}$, $2^{28.9}$, $2^{32.9}$, $2^{27.3}$, $2^{30.8}$, $2^{34.9}$, $2^{27.6}$, $2^{20.3}$ times, respectively, leading to a total complexity of around 2^{35}. On the other hand, when $P_n^*/\overline{P_{n-1}} = 2^{-3}, P_n^*/\overline{P_{n-2}} = 2^{-7}, P_n^*/\overline{P_{n-3}} = 2^{-10}$, then steps 1-8 are repeated $2^{18.7}$, $2^{32.4}$, $2^{36.4}$, $2^{30.8}$, $2^{34.3}$, $2^{38.4}$, $2^{30.9}$, $2^{22.6}$ times, respectively, and hence the complexity is around 2^{39}, while there exist around $2^{22.6}$ (step 8) good 3-round related-key characteristics.

10.2.3 The Split Approach

To build the best n-round characteristic Matsui's approach requires first to build the best characteristics on $1, 2, \ldots, n-1$ rounds because it uses the probabilities of these characteristics. One may be able to skip building the characteristics on some rounds and to assume that they have the same probability as the characteristic on lower number of rounds. Under this assumption, the algorithm still works and finds the best characteristic on n rounds, however the time complexity usually suffers significantly.

Avoiding building all round-reduced characteristics can be done with a different approach. Let us assume we search for characteristic on n rounds that has a probability of at least P_n^*. This n-round characteristic can be seen as a concatenation of two $n/2$-round characteristics, with a combined probability of at least P_n^*. Therefore, one of these two characteristics has a probability at least $\sqrt{P_n^*}$. Indeed we can split the n-round characteristic on any (reasonable) number of k characteristics, each on n/k rounds, and claim that at least one of them has a probability of $\sqrt[k]{P_n^*}$.

Now, let us assume that $n = 3k$, and the n-round characteristic has been split into k three-round characteristics. One of these characteristics (we do not know exactly which), has to have a probability of at least $\sqrt[k]{P_n^*}$. Since it is on three rounds, and it has a bound on its probability, we can use our previous method (observation 2), to build all such characteristics. However, unlike in Matsui's approach, where each of the three rounds has some bound on probability, now we build 3-round characteristics that only have the bound on the combined probability (of all three rounds). Once we have built all such the 3-round characteristics we try to extend them to n rounds (recall that if the difference in the state and in the key is fixed, then it is easy to extend it to more rounds – the difficulty lies in creating all such 3-round characteristics). Interestingly, when extending the three round characteristics, we can use the bounds from Matsui's approach.

For example, let us assume we want to build a characteristic on 9 rounds with a probability at least 2^{-24}. Then we know that one of the three 3-round characteristics has a probability of at least 2^{-8}. First we assume that this is the characteristic on the first three rounds. We build all first 3-round characteristics with probability at least 2^{-8}, i.e. $P_3 \geq 2^{-8}$, and then try to extend them 6 rounds forward, thus obtaining a characteristic on 9 rounds. If we have the probabilities $\overline{P_1}, \ldots, \overline{P_6}$ for the best characteristics on the last 6 rounds, then for rounds 4-9, we can use Matsui's approach, e.g. for 4 rounds we take only those with P_4 such that $P_4 \cdot \overline{P_5} \geq 2^{-24}$, for 5 rounds $P_5 \cdot \overline{P_4} \geq 2^{-24}$, etc. If we do not have the best probabilities than for each round i ($i \geq 4$) we only check if $P_i \geq 2^{-24}$. Then we assume the characteristic on rounds 4-6 has a probability at least 2^{-8}. Again, we build all 3-round characteristics with

at most 2^{-8} and extend them three rounds forward and three backwards (by using Matsui's bounds). Finally, we assume this is the 3-round characteristic on the last three rounds (7-9). We build all such characteristics and extend them 6 rounds backwards (again we can use Matsui's bounds if we have the best probabilities for the first 6 rounds). Among all 9-round characteristics we have produced in these three iterations, we take the one with the highest probability. If such characteristic exist than it is the best characteristic on 9 rounds and it has a probability at least 2^{-24}. If it does not exist then it means all the characteristics on 9 rounds have probability lower than 2^{-24}.

What is the real advantage of this approach compared to related-key Matsui's approach? To find this out, we have to compare the number of possible 3-round related-key characteristic built in the two approaches. In Matsui's algorithm, this number depends on the values $P_n^*/\overline{P_{n-1}}$, $P_n^*/\overline{P_{n-2}}$ and $P_n^*/\overline{P_{n-3}}$, while in the split approach, the number depends only on P_n^*. Hence, when the probabilities $\overline{P_{n-1}}$, $\overline{P_{n-2}}$, $\overline{P_{n-3}}$ are really high, then it is computationally cheaper to build the n-round characteristic with the split approach.

10.3 DES

The notion of (single-key) differentials and differential characteristics was introduced in the seminal paper of Biham and Shamir [22] on cryptanalysis of DES, where the authors presented characteristic on 15 rounds of DES with a probability higher than 2^{-56}. Later in [23], the authors used 13-round characteristic to give the first attack on all 16 rounds of DES. By performing a full search, Matsui [93] has shown that the characteristics found by Biham and Shamir were actually the best round-reduced single-key characteristics for DES. It is well known that S-boxes and the permutation used in the round function of DES are very carefully chosen to avoid single-key differential cryptanalysis and even subtle changes in them can weaken the cipher [23]. Our study of related-key attacks on DES is motivated by the fact that differences in the subkeys could violate some of the design principles and this could lead to new attacks on DES.

We would like to run a full search of the space of all related-key differential characteristics in DES by using the approaches of the previous section. We start with the related-key version of Matsui's algorithm and try to find the best related-key characteristics on as many rounds as possible. Although our search will always find the best characteristics, we should keep in mind that we have a limited computational power. For example, if we try to find the best n-round related-key characteristic that holds with a probability at least P_n^*, then the time complexity of the search mostly depends on the probability $\overline{P_{n-3}}$ of the best characteristics on $(n-3)$ rounds (but also depends on $\overline{P_{n-1}}, \overline{P_{n-2}}$). Our experimental results show that when $P_n^*/\overline{P_{n-3}} < 2^{-12} \sim 2^{-14}$ we do not have the resources to perform the search, hence if for some n this holds, then we will switch to the split approach and continue further with this approach. Note that even in the case of single-key characteristics a similar limitation holds when for some n the ratio $P_n^*/\overline{P_{n-2}}$ is too low.

We start the search by finding the best related-key characteristic on 3 rounds (we assume that $\overline{P_0} = \overline{P_1} = \overline{P_2} = 1$). We fix P_3^* (the probability of the best related-key 3-round characteristic) to 2^{-1} and then gradually decrease by a factor of 2^{-1} if we do not find a characteristic that holds with this probability. There is always a lower bound on this probability – the case of the single-key characteristic (our tool does not make distinction between these two cases, and searches for both). Hence, we can be sure that P_3^* cannot be lower than 2^{-4} (this is the probability of the best single-key characteristic on 3 rounds). Having found the highest P_3^*, we fix $\overline{P_3} = P_3^*$, and then search for $\overline{P_4}$. We fix P_4^* to $\overline{P_3}$, i.e. we assume that the characteristic on 4-rounds has the same probability as the best characteristic on 3 rounds,

and then gradually decrease this probability by a factor 2^{-1} each time when we cannot find 4-round characteristic with such probability. Up to $\overline{P_6}$ we could easily perform the search. However, when searching for $\overline{P_7}$ we could not find anything even when P_7^* was set up to 2^{-18}. We knew that $\overline{P_7}$ could not be lower than $2^{-23.6}$ (the probability of the single-key characteristic on 7 rounds), however if we set $P_7^* = 2^{-23.6}$, then $P_7^*/\overline{P_4} = 2^{-19}$ which is lower than our maximal computational limit of $2^{-12} \sim 2^{-14}$. Therefore, we switched to the split approach for finding the best 7-round related-key characteristic. We started with all possible 3.5-round characteristic (with the first 3.5 rounds and the last 3.5 rounds) with probability of at least 2^{-11} and tried to extend it to 7 rounds, thus we allowed a probability of 2^{-22}. The split approach found that the best related-key characteristic on 7 rounds has a probability of $2^{-20.38}$.

The results of the split search on 7 rounds can be used to find if 8-round characteristic with 2^{-22} exist, which in our case was negative. If we try to apply the related-key Matsui's approach for 8 rounds and allow $P_8^* = 2^{-22}$, then $P_8^*/\overline{P_5} = 2^{-22}/2^{-7.6} = 2^{-14.4}$, which is low. Hence, for 8 rounds we could not use neither Matsui's nor the split approach. However, we noted that the best characteristics of the first 7 rounds have a difference only in a few bits of the key. Hence, we ran a limited search for 8-round characteristic by allowing only a few bit difference in the key. The limited search gave us a characteristic with a probability $2^{-29.75}$ – better than the best single-key characteristic with $2^{30.8}$.

For higher rounds, the related-key Matsui's approach could not work because of the low probabilities ($P_n^*/\overline{P_{n-3}} < 2^{-12} \sim 2^{-14}$). However, if we assume that the 8-round characteristic found by the limited Matsui's approach is the best, then we can still run related-key Matsui's algorithm for the characteristic on 11 rounds. We found that if this holds, then the best related-key characteristics on 11 rounds is the best single-key characteristics.

For finding the best related-key characteristics on 9, 12, and 13 rounds we used our split approach. For 9 rounds, we allowed the 3-round characteristics to have at least $2^{-10.55}$ (because $(2^{-10.55})^3 = 2^{-31.65}$ and the best single-key on 9 rounds has $2^{-31.48}$). The search found that the best 9-round related-key characteristic is the best single-key characteristic. For 12 and 13 rounds, we allowed the starting 3-round characteristics with probability at least $2^{-11.85}$ (because $(2^{-11.85})^4 = 2^{-47.4}$ and the best single-key on 13 rounds has $2^{-47.22}$). Again, we obtained similar results – the best related-key characteristics on 12 and 13 rounds have no difference in the key, i.e. they are the single-key characteristics.

The result for the 13-round[4] related-key characteristic is especially interesting since Biham-Shamir analysis uses it for the attack on the whole DES. This means that *if the attacker uses related-key characteristics, he cannot improve the complexity of Biham-Shamir attack*.

The summary of our findings is presented in Tbl. 10.1. The related-key characteristics for 7 and 8 rounds are given in Fig. 10.1, 10.2.

10.4 DESL

DESL [88] uses a single S-box instead of eight different S-boxes as in DES. This S-box has a special design criteria to discard high probability (single-key) differential characteristics. Indeed, our initial analysis for single-key differential characteristics in DESL confirmed this result. Moreover, we could not find the best single-key differential characteristics (using the original Matsui's tool) for DESL for higher rounds (the absence of the probabilities for the best round-reduced single-key differential characteristics in the submission paper

[4] We rerun the search for characteristics that cover rounds 1 to 12.

Table 10.1: Comparison of the probabilities of the best round-reduced differential single-key and related-key characteristics for DES.

rounds	Single-key	Related-key	Method used
3	$2^{-4.0}$	2^{0}	RK Matsui'
4	$2^{-9.6}$	$2^{-4.61}$	RK Matsui'
5	$2^{-13.21}$	$2^{-7.83}$	RK Matsui'
6	$2^{-19.94}$	$2^{-12.92}$	RK Matsui'
7	$2^{-23.60}$	$2^{-20.38}$	Split
8	$2^{-30.48}$	$2^{-29.75} \leq \overline{P_8} < 2^{-22}$	Limited Matsui'
9	$2^{-31.48}$	$2^{-31.48}$	Split + Matsui'
10	$2^{-38.35}$	$\leq \overline{P_9}$	
11	$2^{-39.35}$	$2^{-39.35}$ if $\overline{P_8} = 2^{-29.75}$	RK Matsui'
12	$2^{-46.22}$	$2^{-46.22}$	Split + Matsui'
13	$2^{-47.22}$	$2^{-47.22}$	Split + Matsui'
14	$2^{-54.09}$	$\leq \overline{P_{13}}$	
15	$2^{-55.09}$	$2^{-55.09}$	RK Matsui'
16	$2^{-61.97}$	$\leq \overline{P_{15}}$	

of DESL [88] seems to confirm our findings). Therefore, even the original Matsui's tool cannot be used (it is infeasible) for finding single-key characteristics, when they hold with low probabilities.

Our related-key Matsui's search algorithm, however, did find the best related-key characteristics for up to 7 rounds. Interestingly, the probabilities of these related-key characteristics are higher in DESL, than in DES (see Tbl. 10.2). For more rounds, we used the split approach as well. Nonetheless, for these characteristics, we were able to find only the upper bounds on their probabilities. For example, for 9-round related-key characteristic we used the split approach with 3-round probability of 2^{-10}. After running the search for the first, middle, and third three rounds, the algorithm did not return any characteristic. This means, there are no related-key characteristics on 9 rounds with probability at least 2^{-30}. Similarly, we used the split approach for finding the upper bound on the probability of the best characteristics for 12-rounds, and the related-key Matsui's approach for the bounds on 10,13, and 15 rounds. Our findings are presented in Tbl. 10.2.

The related-key characteristics that we have found can be used to launch boomerang attacks on the round-reduced cipher. For example, we can launch a related-key boomerang attack on 12 rounds (from round 4 to round 15), with two characteristics on 6 rounds – the first on rounds 4-9, the second on 10-15. The probability of the first characteristic is $2^{-14.68}$ (it is lower because we consider rounds 4-9), while the probability of the second is $2^{-12.09}$. Therefore, the probability of the whole boomerang is $2^{-2\cdot 14.68-2\cdot 12.09} = 2^{-53.54}$.

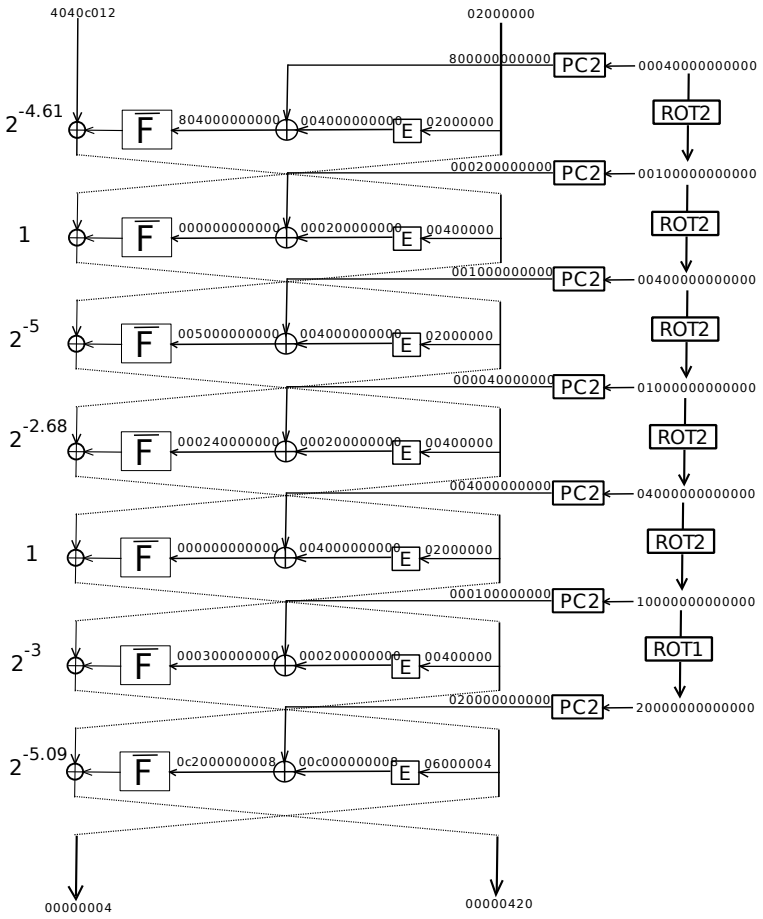

Figure 10.1: The best related-key differential characteristic (with probability $2^{-20.38}$) on the last 7 rounds of DES.

10.5 s^2DES

Another variant of DES called s^2DES was proposed in [82]. The search for the best single-key differential characteristics in s^2DES was performed in [135]. For this purpose the authors used Matsui's tool. This analysis showed that the best round-reduced differential characteristics in s^2DES have higher probabilities than in DES.

We ran our search for related-key characteristics using only our related-key approached based on Matsui's algorithm. We noted that for each single-key characteristic on n-rounds, the value $\overline{P_n}/\overline{P_{n-3}}$ is at least $2^{-12.75}$ (for $n = 8$, see Tbl. 10.3), hence building all 3-round related-key characteristic might be feasible. However, the values $\overline{P_{n-3}}$ for different n could be updated, because they were the probabilities in the single-key scenario (the probability in the related-key scenario is not less than in the single-key). Indeed, the probabilities of

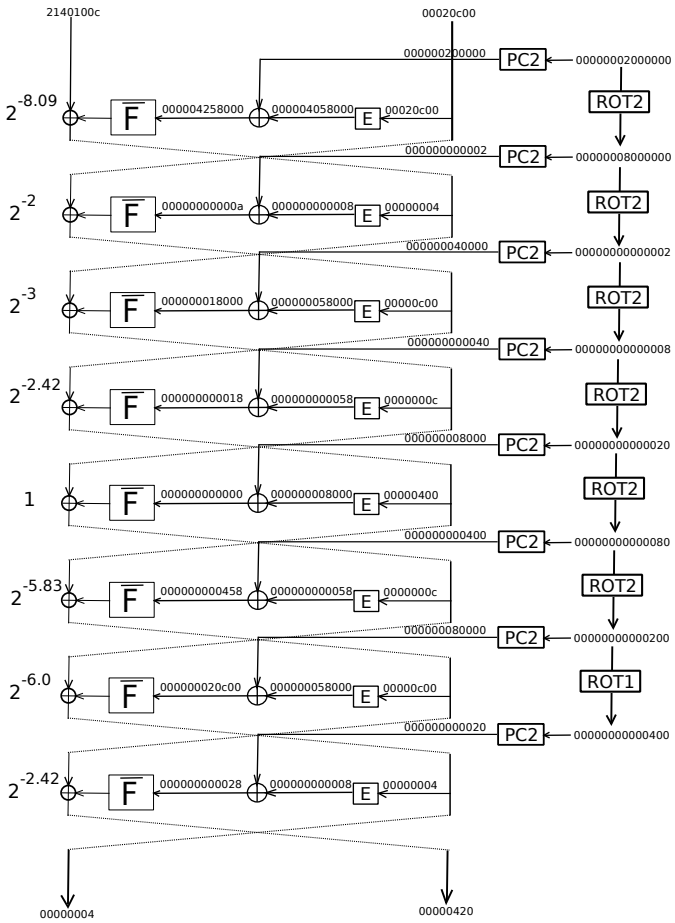

Figure 10.2: Related-key differential characteristic (with probability $2^{-29.75}$) on the last 8 rounds of DES.

the round-reduced related-key characteristics for the first 6 rounds, were higher than the probabilities of the single-key characteristics. This made $\overline{P_5}$ to be 2^{-8} instead of $2^{-9.22}$ as in the single-key case. Hence, for the related-key characteristic on 8 rounds, we had to allow $\overline{P_8}/\overline{P_5} = 2^{-22}/2^{-8} = 2^{-14}$ for the active S-boxes in the three rounds, instead of the previous $2^{-12.75}$. However, we were able to perform the search for this 7-round characteristic but with a significant computational cost – the search took around 3 weeks on 64 CPU cores.

After the sixth round, we found that all the best related-key characteristics have the same probability as the single-key (indeed they are single-key). The probabilities of the best single and related-key round-reduced characteristics are given in Tbl. 10.3.

mbox

Table 10.2: Probabilities of the best round-reduced related-key differential characteristics for DESL.

Round	Probability
3	2^{0}
4	$2^{-4.67}$
5	$2^{-7.24}$
6	$2^{-12.09}$
7	$2^{-19.95}$
8	$\leq \overline{P_7}$
9	$< 2^{-30}$
10	$< 2^{-31}$
11	$\leq \overline{P_{10}}$
12	$< 2^{-40}$
13	$< 2^{-41}$
14	$\leq \overline{P_{13}}$
15	$< 2^{-50}$
16	$< 2^{-51}$

Table 10.3: Comparison of the probabilities of the best round-reduce differential single-key and related-key characteristics for s^2DES.

rounds	Single-key	Related-key
3	$2^{-4.39}$	2^{0}
4	$2^{-6.8}$	$2^{-5.19}$
5	$2^{-9.22}$	$2^{-8.0}$
6	$2^{-14.35}$	$2^{-12.61}$
7	$2^{-17.03}$	$2^{-17.03}$
8	$2^{-21.96}$	$2^{-21.96}$
9	$2^{-22.71}$	$2^{-22.71}$
10	$2^{-27.35}$	$2^{-27.35}$
11	$2^{-28.39}$	$2^{-28.39}$
12	$2^{-34.07}$	$2^{-34.07}$
13	$2^{-34.07}$	$2^{-34.07}$
14	$2^{-39.75}$	$2^{-39.75}$
15	$2^{-39.75}$	$2^{-39.75}$
16	$2^{-45.42}$	$2^{-45.42}$

Part V

Distinguishers and Preimage Attacks on Hash Functions

The last contribution is devoted to rotational attacks on cryptographic hash functions and ciphers, and meet-in-the-middle attacks on cryptographic hash functions. The rotational attacks can be seen as distinguishers for the primitives (usually compression functions, or the block ciphers in the open-key/secret-key mode). On the other hand, the meet-in-the-middle attacks are exploited to launch preimage attacks on the whole hash function constructions. The presented attacks are on SHA-3 candidates Skein (and the underlying cipher Threefish), Blue Midnight Wish, Boole, Edon-R and Sarmal. The analysis given here is available in the following papers:

- [77] **Rotational Cryptanalysis of ARX**, FSE 2010
- [78] **Rotational Rebound Attacks on Reduced Skein**, ASIACRYPT 2010
- [113] **Rotational Cryptanalysis of (Modified) Versions of BMW and SIMD**, unpublished
- [80] **Meet-in-the-Middle Attacks on SHA-3 Candidates**, FSE 2009

Chapter 11

Rotational Distinguishers

Rotational analysis is a relatively new type of attack. In differential analysis, for a pair of inputs (X, Y), the adversary follows the propagation of the difference $X \oplus Y$. On the other hand, in rotational analysis, the adversary examines the propagation of a rotational pair of inputs $(X, X \lll_r)$.

A pair of n-bit words (X, Y) can be fully rotational, i.e. $X \lll_r \oplus Y = 0$, or only in t bits, i.e. $h_w(X \lll_r \oplus Y) = n - t$, where h_w is the Hamming weight function. Basically, we require the output pairs of all internal transformations to be fully rotational. We make an exception for the last transformation, where it is enough to have t rotational output bits as they can be used to build a distinguisher.

For some transformation, instead of taking rotational input pairs, it is better to introduce correction by XOR-ing (or adding modularly) some low Hamming weight word to the second input, i.e. instead of $(X, X \lll_r)$, we take $(X, X \lll_r \oplus \delta)$. If X is input to another transformation, then the correction most likely has to be canceled (often by XOR-ing the same correction to some other input). Otherwise, a non-rotational input pair may significantly decrease the probability of a rotational output pair for the second transformation.

Since most of the transformations preserve the rotational property only with some probability, we can observe errors in the cases when the property does not hold. A rotational error e_F of a transformation F is defined as $e_F = F(X) \lll_r -F(X \lll_r)$. Depending on the actual value of X, different values for the rotational error may be produced. The errors may cancel each other as well. For example, let the output pairs of two distinct transformations have the same rotational errors, but with opposite sign, i.e. $F_1(X) \lll_r -F_1(X \lll_r) = e, F_2(Y) \lll_r -F_2(Y \lll_r) = -e$. If the outputs of F_1, F_2 are inputs to an addition, then the output pair of the addition will be rotational (with the rotational probability of addition), since $(F_1(X) + F_2(Y)) \lll_r = F_1(X) \lll_r + F_2(X) \lll_r = F_1(X \lll_r) + e + F_2(Y \lll_r) - e = F_1(X \lll_r) + F_2(Y \lll_r)$.

The primitives we analyze have only a reduced set of operations – XORs, additions, substractions, rotations, and shifts. The rotational probabilities p_F of these transformations are given in the following lemmas:

Lemma 5 (Addition,[46]) *For n-bit words x, y, and a positive integer r*

$$\mathbf{P}((x + y) \lll_r = x \lll_r + y \lll_r) = \frac{1}{4}(1 + 2^{r-n} + 2^{-r} + 2^{-n}).$$

Lemma 6 (Rotation) *For n-bit word x and positive integers r, r'*

$$\mathbf{P}((x \lll_r) \lll_{r'} = (x \lll_{r'}) \lll_r) = 1.$$

Lemma 7 (XOR) *For n-bit words x, y, and a positive integer r*

$$\mathbf{P}((x \oplus y) \lll_r = x \lll_r \oplus y \lll_r) = 1.$$

Lemma 8 (Subtraction,[46]) *Given a pair of n-bit words x, y and a positive integer r, then*

$$\mathbb{P}((x - y) \lll_r = x \lll_r - y \lll_r) = \frac{1}{4}(1 + 2^{r-n} + 2^{-r} + 2^{-n}).$$

Lemma 9 (Shifts,[113]) *Given an n-bit word x and two positive integers r, s, then*

$$\mathbb{P}((x \ll_s) \lll_r = (x \lll_r) \ll_s) = 2^{-2t},$$

$$\mathbb{P}((x \gg_s) \lll_r = (x \lll_r) \gg_s) = 2^{-2t},$$

where $t = min(r, s, n - r, n - s)$.

Lemma 10 (Boolean function,[113]) *Given a bitwise Boolean function f, then*

$$\mathbb{P}(f(x) \lll_r = f(x \lll_r)) = 1,$$

where x is a n-bit word and r is some positive integer.

Now let us focus on multi additions and multi substractions. For this purpose let define $N_k(i, t)$ as:

$$N_k(i, t) = \sum_{j=0}^{\lfloor \frac{i}{t+1} \rfloor} (-1)^j \binom{k}{j} \binom{i - j(t+1) + k - 1}{i - j(t+1)}.$$

Then the following lemmas hold.

Lemma 11 (Multi additions,[113]) *Given n-bit words $x_1, \ldots, x_k$ and a positive integer r, then*

$$\begin{aligned}&\mathbb{P}((x_1 + \ldots + x_k) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r) = \\ &= \frac{1}{2^{nk}} \sum_{j=0}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} N_{k+1}(j2^n + 2^r - 1, 2^r - 1) \cdot \sum_{j=0}^{\lfloor \frac{k-1}{2^r} \rfloor} N_{k+1}(j2^n + 2^{n-r} - 1, 2^{n-r} - 1).\end{aligned}$$

Lemma 12 (Multi additions and substractions,[113]) *Given n-bit words $x_1, \ldots, x_k, y_1, \ldots, y_l$ and a positive integer r, then*

$$\begin{aligned}&\mathbb{P}((x_1 + \ldots + x_k - y_1 - \ldots - y_l) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r - y_1 \lll_r - \ldots - y_l \lll_r) = \\ &= \frac{1}{2^{n(k+l)}} \sum_{j=-\lceil \frac{l}{2^{n-r}} \rceil}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} N_{k+l+1}(j2^n + (l+1)(2^r - 1), 2^r - 1) \cdot \sum_{j=-\lceil \frac{l}{2^r} \rceil}^{\lfloor \frac{k-1}{2^r} \rfloor} N_{k+l+1}(j2^n + (l+1)(2^{n-r} - 1), 2^{n-r} - 1).\end{aligned}$$

The use of constants, which may not form a rotational pair, does not restrain our analysis, but makes it more sophisticated. An addition of a constant may generate a rotational error (the exact probability depends on r, the constant value, and which type of addition is used — modular or XOR). On the other hand, a modular addition of variables also may generate an error, and with some probability these errors compensate each other. The probability is higher if the constant has low Hamming weight and if its positions of ones (in the binary representation) are concentrated close to the positions where addition errors appear. It may also happen that a constant is added by XOR and is invariant of the rotation: $C = C \lll_r$. Then the rotation property passes the addition of a constant for free. The subkey indices in Threefish are examples of low-weight constants. However, they are not compensated by a single addition, only by two previous additions, which leads to an error in the adjacent key addition.

11.1 Rotational Cryptanalysis of Threefish and Skein

In this section we attack the block cipher Threefish and the compression function of Skein with rotational cryptanalysis. We demonstrate that a rotational pair of Threefish ciphertexts can be obtained faster than for a random permutation, which provides both a distinguisher and a key recovery attack. A similar result is obtained for Skein.

11.1.1 Specification of Threefish and Skein

Skein is a family of hash functions, based on the block cipher Threefish of which the following versions are relevant for the SHA-3 proposal: Threefish-256 — 256-bit block cipher with 256-bit key and Threefish-512 — 512-bit block and key. The cipher family Threefish has one more member – Threefish-1024 with 1024-bit block and key. Both the internal state I and the key K consist of N_w ($N_w = 4, 8, 16$ for Threefish-256,-512,-1024, respectively) 64-bit words. The N_w words of the s-th subkey K^s are defined as follows:

$$\begin{aligned} K^s_j &= K_{(s+j) \bmod (N_w+1)}, \quad 0 \le j \le N_w - 4; \\ K^s_{N_w-3} &= K_{(s+N_w-3) \bmod (N_w+1)} + t_{s \bmod 3}; \\ K^s_{N_w-2} &= K_{(s+N_w-2) \bmod (N_w+1)} + t_{(s+1) \bmod 3}; \\ K^s_{N_w-1} &= K_{(s+N_w-1) \bmod (N_w+1)} + s, \end{aligned}$$

where s is a round counter, t_0 and t_1 are tweak words, and

$$t_2 = t_0 + t_1, \quad K_{N_w} = \lfloor 2^{64}/3 \rfloor \oplus \bigoplus_{j=0}^{N_w-1} K_j.$$

The formal description of internal rounds is as follows. Let N_r be the number of rounds ($N_r = 72$ for Threefish-256,-512, and $N_r = 80$ for Threefish-1024). Then for every $1 \le d \le N_r$

- If $d \bmod 4 = 1$ add a subkey by setting $I_j \leftarrow I_j + K^{d/4}_j$;
- For $0 \le j < N_w/2$ set $(I_{2j}, I_{2j+1}) \leftarrow \mathrm{MIX}((I_{2j}, I_{2j+1}))$;
- Apply the permutation π on the state words.

At the end, a subkey $K^{N_r/4}$ is added. The operation MIX has two inputs x_0, x_1 and produces two outputs y_0, y_1 with the following transformation:

$$y_0 = x_0 + x_1$$
$$y_1 = (x_1 \lll_{R_{(d \bmod 8)+1,j}}) \oplus y_0$$

The exact values of the rotation constants $R_{i,j}$ as well the permutations π (which are different for each version of Threefish) can be found in [53].

The compression function $C(H_{i-1}, M_i)$ of Skein is defined as:

$$C(H_{i-1}, M_i) = E_{H_{i-1},T_i}(M_i) \oplus M_i,$$

where $E_{K,T}(P)$ is the Threefish cipher, H_{i-1} is the previous chaining value, T_i is the tweak, and M_i is the message block.

11.1.2 Attacks on Simplified Versions of Threefish

There are two places in the key schedule of Threefish where we encounter constants: 1) K_{N_w} is obtained with a XOR of all key words and the constant $C_5 = \lfloor 2^{64}/3 \rfloor$, and 2) the last subkey word $K^s_{N_w-1}$ has a modular addition of the round counter s. Hence, in addition to the original Threefish, we can obtain three simplified versions by discarding these constant XOR and counter additions. Our attacks are in the related-key scenario, where all the key and plaintext words compose rotational pairs, i.e. if the first key and the plaintext have the values $(k_0, \ldots, k_{N_w}), (p_0, \ldots, p_{N_w-1})$ then the second (related) key and the plaintext have the values $(k_0 \lll_r, \ldots, k_{N_w} \lll_r), (p_0 \lll_r, \ldots, p_{N_w-1} \lll_r)$.

The simplest version of Threefish is without the XOR of C_5 and the additions of the round counters. We can fix the rotation amount in the rotational pair to 1 in order to get the best probability — $2^{-1.415}$ per addition. A simple MIX has only one addition, hence a round of Threefish-256 has only two additions. The 59-round version of Threefish-256 has $2 \cdot 59 = 118$ additions in the MIX of the rounds and $4 \cdot 15 = 60$ additions of the subkey words, so the probability that a rotational pair of key/plaintext (with a rotation equal to 1) will produce a rotational pair of ciphertexts is $2^{-1.415 \cdot (118+60)} = 2^{-252}$, which is higher than for a random permutation. Every right pair also provides information on leftmost key bits of each key word, so we get a valid key recovery attack with a complexity of about 2^{252} encryptions. The same reasoning is applicable for 59-round distinguishers for Threefish-512 which has a complexity of 2^{504} and to Threefish-1024 and 2^{1008}, because these ciphers differ from Threefish-256 only in the size of the state and the key.

When the XOR of C_5 is present, then the only difference is that we cannot use the rotation amount 1 because $C_5 \lll 1 \neq C_5$, i.e. the constant C_5 is not invariant of rotation 1. Instead we can use rotation 2, and get attacks on 50 rounds. The complexity of the attack on Threefish-256 is $2^{1.67 \cdot (2 \cdot 50 + 4 \cdot 13)} = 2^{253.8}$. For Threefish-512 and Threefish-1024 they are $2^{507.6}$ and $2^{1015.2}$.

For the version of Threefish without the constant C_5 and the round counters, we get much better results if we consider a weak key class, for which it is unlikely to get errors during the modular addition. Let the three leftmost bits of each key word be zero, and consider rotation to the left by one bit. Then the rotational probability of addition is equal to $2^{-0.28}$, and therefore the total probability for the full 72-round Threefish-256, the version without C_5 and round counters, is $2^{-1.415 \cdot 2 \cdot 72 - 0.28 \cdot 4 \cdot 18} = 2^{-224}$. The size of the weak key class that we attack is $2^{61 \cdot 4} = 2^{244}$, so we get a valid attack on a very large key class. Analogously, we can attack a weak key class with 2^{488} keys of Threefish-512 with complexity 2^{448}, and Threefish-1024 with a complexity 2^{950} (the complexity is slightly higher because Threefish-1024 has 80 rounds).

11.1.3 Attacks on Threefish by XORing Corrections

Let us try to apply rotational analysis to the original version of Threefish. This means we have to deal with the round counters – low weight constants. In order to bypass them we introduce corrections in the key pair. Let K be the first secret key. Then the second key K' is defined as follows:

$$K'_i = K_i \lll_r \oplus e_i$$

The use of rotational pairs with corrections is illustrated in Fig. 11.1.

We have found experimentally, that the values of the corrections e_i should not be larger than 16 (otherwise they do not cancel the round counters). For Threefish-256 and Threefish-512 it is feasible to find by brute force the exact values for the corrections that cancel the counters with maximal probability. For Threefish-1024 we took the values that were good in Threefish-512.

The corrections forbid to obtain clear formula for the probability of addition of a rotational pair. Hence, we have found these probabilities empirically. We have grouped two rounds with a subkey addition (round – subkey addition – round), and by Monte Carlo method found the probability that a rotational pair of states at the input of these two rounds and a rotational pair of subkeys with corrections will produce a rotational pair of states at the output. Based on these values, we have produced the probabilities of the best round-reduced rotational pairs. The explicit round-by-round values of the probabilities are given in Tbl. 11.3. The results are given for the original versions as well as for the versions without the C_5 (except in the case for Threefish-1024 where the probability of the version without C_5 is lower than for the original Threefish-1024).

We can break 39, 42, and 43.5[1] rounds of the original versions of Threefish-256,-512, and -1024 with complexity of $2^{252.4}, 2^{507}, 2^{1014.5}$ encryptions respectively. The attacks procedures follow the same algorithm:

1. Generate a random plaintext P and encrypt it on K;
2. Compute P' and encrypt on K';
3. Check whether $(E_K(P), E_{K'}(P'))$ is a rotational pair.

A rotational pair discloses information about leftmost key bits of every key word. The plaintext P' is computed by the following rule:

$$P'_i = P_i \lll_r \oplus d_i.$$

The plaintext and the key corrections are defined separately for all the three versions of Threefish in Tbl. 11.1.

For the versions with counters but without C_5, again we only change the rotation amount to 1, and obtain attacks on 44 and 51.5 rounds of Threefish-256, -512 with a complexity of 2^{252} and $2^{506.7}$ encryptions.

11.1.3.1 On the Probabilities of Rotational Characteristics for Threefish

One part of the probabilities of the trails presented at Tbl. 11.3 is computed theoretically, and one part practically.

When the rotational pair does not have corrections, then we use the probability of addition defined by Lemma 5. In the original versions (with C_5) the rotation amount is 2, so the

[1] .5 means without the last subkey addition.

Table 11.1: Corrections in the plaintext pairs (d_i) and the key pairs (e_i) in Threefish.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Threefish-256																
d_i	3	10	3	15												
e_i	6	10	6	15												
Threefish-512																
d_i	0	6	3	6	3	6	3	6								
e_i	5	6	6	6	6	6	6	6								
Threefish-1024																
d_i	0	6	3	6	3	6	3	6	3	6	3	6	3	6	3	6
e_i	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6

probability of addition is $2^{-1.676}$. When C_5 is absent then we use a rotation amount 1, and the probability of addition becomes $2^{-1.415}$. Two consecutive rounds of Threefish-256 have 4 MIX and each MIX has one addition. Hence, two rounds with no subkey additions have a probability of $2^{-6.6}$. Analogously, for Threefish-512 and Threefish-1024 we get $2^{-13.3}$ and $2^{-26.6}$, respectively. These numbers translate into $2^{-5.7}, 2^{-11.3}, 2^{-22.6}$ for the versions without C_5.

When there are corrections in the rotational pair, we find the probabilities of two rounds (one round + subkey addition + one round) experimentally. The probabilities for the round 1 (key addition + one regular round) are also computed experimentally.

We have used to following corrections:

- for Threefish-256 without C_5 in the key: 7, 2, 2, 6; in the plaintext: 2,2,7,6;
- for Threefish-512 without C_5 in the key: 2, 1, 3, 1, 7, 1, 7, 3; in the plaintext: 7,1,6,1,2,1,2,3

11.1.4 Attacks on Skein by Adding Corrections

We follow the idea of the previous section, and introduce corrections in the Threefish keys. But unlike in the previous attack, we consider modular corrections, i.e. we define the related-key pair by (K_A, K_B), where $K_B = K_A \lll_2 + e$, e is a low-weight correction, "+" is modular addition and the rotation amount is fixed to 2. Each 64-bit word w in Skein can be seen as a concatenation of two words w_1, w_2, i.e. $w = w_1 || w_2$ where w_1 represent the two most significant bits of w and w_2 the rest 62 bits.

To obtain a high number of rounds in the outbound phase, we carefully choose optimal corrections and fix some of the key bits. More specifically, we found the best values of key bits with the optimized exhaustive search. Now we explain how to optimize the search in Skein-256 (Figure 11.1.4).

We consider two rounds of Skein-256 with a subkey addition in between (rounds 4-5, 8-9, etc.). Note that the outer double rounds (6-7, 10-11, etc) simply keep the rotational pairs,

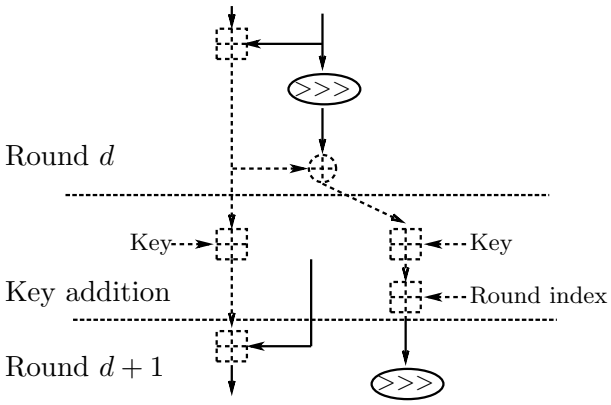

Figure 11.1: Rotational corrections in the key addition layer of Threefish. Dashed lines contain rotational pairs with errors.

Table 11.2: Attack complexities for different versions of Threefish; the weak key class on full-round Threefish.

Cipher	Round index	Constant $\lfloor 2^{64}/3 \rfloor$	Rounds	Complexity
Threefish-256	no	no	59	2^{252}
	no	yes	50	$2^{253.8}$
	yes	no	44	$2^{251.4}$
	yes	**yes**	39	$2^{254.1}$
(weak key of 2^{244})	no	no	72	2^{224}
Threefish-512	no	no	59	2^{504}
	no	yes	50	$2^{507.6}$
	yes	no	51.5	$2^{505.5}$
	yes	**yes**	42	2^{507}
(weak key of 2^{488})	no	no	72	2^{448}
Threefish-1024	no	no	59	2^{1008}
	no	yes	50	$2^{1015.2}$
	yes	**yes**	43.5	$2^{1014.5}$
(weak key of 2^{976})	no	no	80	2^{950}

Table 11.3: Probabilities for the rotational pairs of different versions of Threefish.

Rounds	Threefish-256		Threefish-512		Threefish-1024
	original	without C_5	original	without C_5	original
1	-13.1	-10.5	-22.8	-21.6	-45.6
$2-3$	-6.6	-5.7	-13.3	-11.3	-26.7
$4-5$	-17.57	-12.56	-29.47	-25.92	-61.48
$6-7$	-6.6	-5.7	-13.3	-11.3	-26.7
$8-9$	-15.33	-13.95	-31.33	-22.68	-63.32
$10-11$	-6.6	-5.7	-13.3	-11.3	-26.7
$12-13$	-15.60	-12.05	-29.73	-27.99	-61.68
$14-15$	-6.6	-5.7	-13.3	-11.3	-26.7
$16-17$	-21.08	-14.17	-34.35	-25.81	-66.44
$18-19$	-6.6	-5.7	-13.3	-11.3	-26.7
$20-21$	-18.46	-14.6	-37.25	-29.43	-68.82
$22-23$	-6.6	-5.7	-13.3	-11.3	-26.7
$24-25$	-21.47	-17.41	-34.38	-26.89	-66.34
$26-27$	-6.6	-5.7	-13.3	-11.3	-26.7
$28-29$	-21.55	-13.44	-36	-25.61	-67.31
$30-31$	-6.6	-5.7	-13.3	-11.3	-26.7
$32-33$	-21.74	-16.64	-37.63	-26.74	-69.28
$34-35$	-6.6	-5.7	-13.3	-11.3	-26.7
$36-37$	-22.96	-17	-38.17	-25.12	-67.79
$38-39$	-6.6	-5.7	-13.3	-11.3	-26.7
$40-41$		-17.74	-36.24	-31.34	-69.64
$42-43$		-5.7	-6.6	-11.3	-26.7
$44-45$		-18.89		-30.60	-13.3
$46-47$		-5.7		-11.3	
$48-49$		-2.8		-33.19	
$50-51$				-11.3	
52				-5.7	
Total rounds	39	44	42	51.5	43.5
Total probability	-254.1	-251.4	-507	-505.5	-1014.5

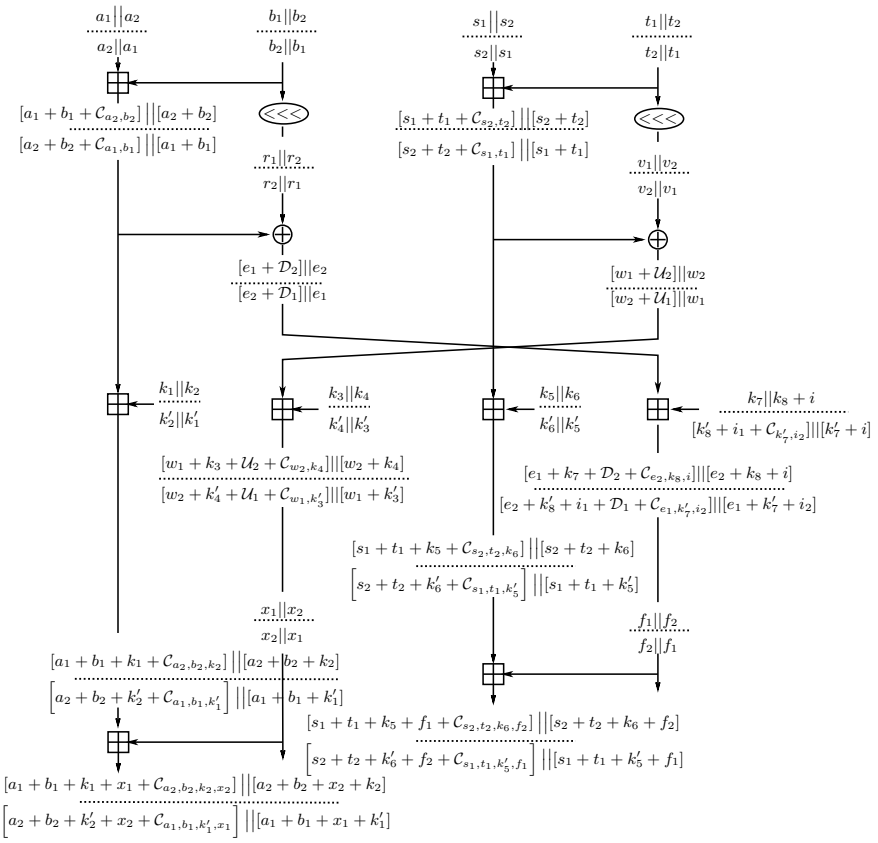

Figure 11.2: Rotational pair through two rounds with key addition of Skein-256.

so the probability does not depend on the number of round. The outer rounds probability is $2^{-8.5}$ for Skein-256 and 2^{-17} for Skein-512.

We denote the four words of the internal state before the double rounds by (A, B, C, D). Therefore, we have

$$(A, B, C, D) = (a_1||a_2,\ b_1||b_2,\ s_1||s_2,\ t_1||t_2);$$
$$(A \lll_2, B \lll_2, C \lll_2, D \lll_2) = (a_2||a_1,\ b_2||b_1,\ s_2||s_1,\ t_2||t_1).$$

Similarly, we denote by

$$K_A = [k_1||k_2,\ k_3||k_4,\ k_5||k_6,\ k_7||k_8]; \quad K_B = [k_2'||k_1',\ k_4'||k_3',\ k_6'||k_5',\ k_8'||k_7'].$$

the rotational pair of subkeys. Then the corrections e_i can be defined as

$$e_i = k'_{2i+1}||k'_{2i+2} - k_{2i+1}||k_{2i+2}.$$

In Figure 11.1.4 the pairs are presented one a top of another with the symbol "- - - - - - -" between them. By $C_{z_1,\ldots,z_k}$ we denote the carry from the sum $z_1 + \ldots + z_k$, i.e. when $z_i < 2^r$,

then $C_{z_1,\ldots,z_k} = (z_1 + \ldots + z_k) \ggg_r$. The variables $r, v, \mathscr{D}, \mathscr{U}, x, f$ are introduced to maintain the 2+62 bit representation of the words. With $i = i_1||i_2$ we denote the round counter. Since the rotation preserves the rotational property, we can omit the rotations in the second round of the double subkey rounds, and only require rotational output pairs after the additions in this round. To obtain such pairs for the first output, the following conditions have to hold:

$$\begin{aligned} a_1 + b_1 + k_1 + x_1 + C_{a_2,b_2,k_2,x_2} &= a_1 + b_1 + x_1 + k_1^{'} \\ a_2 + b_2 + x_2 + k_2 &= a_2 + b_2 + k_2' + x_2 + C_{a_1,b_1,k_1',x_1} \end{aligned}$$

Similarly, for the rest 3 outputs, we get the following conditions:

$$\begin{aligned} w_1 + k_3 + \mathscr{U}_2 + C_{w_2,k_4} &= w_1 + k_3' \\ w_2 + k_4 &= w_2 + k_4' + \mathscr{U}_1 + C_{w_1,k_3'} \\ s_1 + t_1 + k_5 + f_1 + C_{s_2,t_2,k_6,f_2} &= s_1 + t_1 + k_5' + f_1 \\ s_2 + t_2 + k_6 + f_2 &= s_2 + t_2 + k_6' + f_2 + C_{s_1,t_1,k_5',f_1} \\ e_1 + k_7 + \mathscr{D}_2 + C_{e_2,k_8,i} &= e_1 + k_7' + i_2 \\ e_2 + k_8 + i &= e_2 + k_8' + i_1 + \mathscr{D}_1 + C_{e_1,k_7',i_2} \end{aligned}$$

The above 8 equations, can be reduced to:

$$k_1' - k_1 = C_{a_2,b_2,k_2,x_2} \tag{11.1}$$
$$k_2' - k_2 = -C_{a_1,b_1,k_1',x_1} \tag{11.2}$$
$$k_3' - k_3 = C_{w_2,k_4} + \mathscr{U}_2 \tag{11.3}$$
$$k_4' - k_4 = -(C_{w_1,k_3'} + \mathscr{U}_1) \tag{11.4}$$
$$k_5' - k_5 = C_{s_2,t_2,k_6,f_2} \tag{11.5}$$
$$k_6' - k_6 = -C_{s_1,t_1,k_5',f_1} \tag{11.6}$$
$$k_7' - k_7 = C_{e_2,k_8,i} + \mathscr{D}_2 - i_2 \tag{11.7}$$
$$k_8' - k_8 = i - i_1 - (C_{e_1,k_7',i_2} + \mathscr{D}_1) \tag{11.8}$$

This system gives as a hint how to choose the corrections e_i and the values of some of the subkey bits. For each carry $C_{z_1,\ldots,z_k}$ it holds $0 \leq C_{z_1,\ldots,z_k} < k$. Yet the probability that a carry will take a specific value in this range, when z_i are randomly chosen, is not uniformly distributed. When the carries come from sums with 4 terms, the probability is highest for the values 1 and 2. Therefore, for our brute force, we limit the differences $k_1' - k_1, k_2 - k_2', k_5' - k_5, k_6' - k_6$, only to these two values.

The variables $\mathscr{U}_1, \mathscr{U}_2, \mathscr{D}_1, \mathscr{D}_2$, are determined as follows:

$$\begin{aligned} \mathscr{U}_1 &= ((s_2 + t_2 + C_{s_1,t_1}) \oplus v_2) - ((s_2 + t_2) \oplus v_2) \\ \mathscr{U}_2 &= ((s_1 + t_1 + C_{s_2,t_2}) \oplus v_2) - ((s_2 + t_2) \oplus v_2) \\ \mathscr{D}_1 &= ((a_2 + b_2 + C_{a_1,b_1}) \oplus r_2) - ((a_2 + b_2) \oplus r_2) \\ \mathscr{D}_2 &= ((a_1 + b_1 + C_{a_2,b_2}) \oplus r_1) - ((a_1 + b_1) \oplus r_1) \end{aligned}$$

These variables can take only odd values and a zero. Since C_{w_2,k_4} can take $0, 1$ and $\mathscr{U}_2$ can take $0, 1$ it means that $k_3' - k_3$ (see (11.3)) can also take 1 and 2 (the same values as the one for the subkeys discussed above). A similar reasoning is applicable to the difference $k_4 - k_4'$. The differences $k_7' - k_7$, $k_8 - k_8'$ that are left, are the only one that actually depend

on the round counter. Yet, since $C_{e_2,k_8,i}$ can take the values $0, 1$[2], i.e. it is not fixed but rather flexible, the whole expression $C_{e_2,k_8,i} + \mathscr{D}_2 - i_2$, for any i_2 can take the values $1, 2$ (recall that $\mathscr{D}_2$ can be any odd value). Therefore the difference $k'_7 - k_7$ can be 1 or 2 (with probability that depends on the round counter i_2). Finally, let us focus on the difference $k'_8 - k_8$ which is determined by the expression $i - i_1 - C_{e_1,k'_7,i_2} - \mathscr{D}_1$. For a specific counter i, when $k'_7 + e_2 = 0$, the carry C_{e_1,k'_7,i_2} is fixed. Hence in this case, the whole expression can take only one value, 1 or 2, but not the both. This limits $k'_8 - k_8$ to only a single value.

Now recall that k_i, k'_i are the values of the particular subkey words, and not the key words. Once we fix all of the differences in the subkey words of some round, then in the next round, practically the same differences will appear shifted by one index. Also, since the value of the difference in the last key word K_4 is determined from the other words, we would have to fix the values of k_1, k_3, k_5, k_7 and the two least significant bits of k_2, k_4, k_6, k_8 so that the difference in K_4 will be as expected. We fix only two bits because we choose the initial difference to be 1 or 2.

In our brute force search, first we find good values for the differences and the two most significant key bits of each key word. We try all possible differences 1 or 2, and then we fix the key bits values, such that the difference in the two most significant bits of K_4 will also be 1 or 2, and we take into account the limitation on $k'_8 - k_8$ for each counter. Then, we try all possible differences 1 and 2 in the least 62 bits of the each key word. We choose the differences that pass with highest probability through the double subkey rounds. Also, we fix the 2 least significant bits in each key word, so that the difference in the least 62 bits of K_4 will also be 1 or 2. Finally, to increase the probability we fix the values of the bits 60,61 (the next two bits after the 2 most significant bits). This results in fixing the two most significant bits of k_2, k_4, k_6, k_8 which in return increases the probability that the carries take the expected values.

Rather than finding the above values through a theoretically small brute force, we have tested our approach on a real double subkey rounds Skein-256. That is, most of the values, were found and confirmed to be good by taking rotational input pairs of states and rotational input pair of key words with corrections and testing the probabilities on double subkey rounds. In some cases the theoretical probabilities did not coincide with the empirical. This is because there are some hidden dependencies. For example, both $\mathscr{U}_1$ and $k'_5 - k_5$ depend on s_2, t_2. Once we had the optimal corrections (and some bit values) of the keys for the double subkey rounds, we found the probability for 4 consecutive rounds. We start with a random rotational input pair of states and go through three rounds. Then we add the subkeys (with the particular counters) and then we go for an additional round.

We fix 6 bits in K_A: 4 MSBs and 2 LSBs, and 6 bits in K_B: 2 MSBs and 4 LSBs. The values of these bits are given at Table 11.4. In Skein-256 the probability to pass rounds 3–42 (i.e. 10 key additions) is 2^{-244}. Round-by-round probabilities are given at Table 11.5.

Optimal values for the differences and some key bits can be obtained for Skein-512 as well. A property of the double subkey rounds Skein-512 that helps to run the brute force search is that these two double subkey rounds can be split into two non-intercepting halves (see Fig.11.3). Then, for each half, the optimal differences can be found independently. Note that this simply speeds up the brute force for optimal differences and values, but has no impact on the actual probability of the inbound phase. Unlike Skein-256, in Skein-512 we could not find empirically the probabilities for 4 consecutive rounds because they were too low. Hence, we considered each 4 rounds as double round + double subkey round and simply multiplied the probabilities of these two. The values for the optimal 6 bits of each

[2]It can take the value 2 as well, but the probability is really low because the counter i is only 4-5 bits.

Table 11.4: Pre-fixed values of key bits in Skein-256. The middle 58 bits of k_i coincide (with regard to the rotation) in K_A and K_B.

	k_0	k_1	k_2	k_3	k_4
K_A	0111..10	0100..11	0011..10	0000..11	0101..01
K_B	11..0011	00..1010	11..0110	00..1001	01..0011

Table 11.5: Round-by-round rotational probabilities for Skein-256

Rounds	1-2	3-5	6-9	10-13	14-17	18-21
Prob. log_2	–	−15.13	−21.97	−21.84	−24.44	−24.69
Rounds	22-25	26-29	30-33	34-37	38-41	42
Prob. log_2	−23.83	−26.09	−23.44	−31.75	−27.09	−3.3

key word in Skein-512 are given in Table 11.6. In Skein-512 the probability to pass rounds 3–46 is about 2^{-494} (details in Table 11.7).

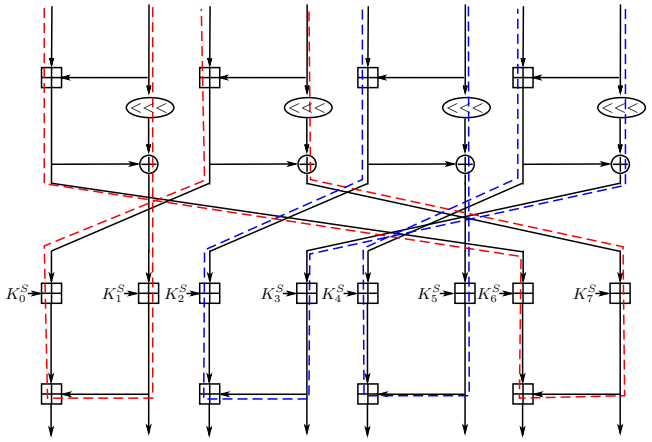

Figure 11.3: Double subkey round in Skein-512 divided into two nonintersecting halves – red and blue.

Table 11.6: Pre-fixed values of key bits in Skein-512

	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
K_A	0111..01	0100..01	0011..01	0000..01	0111..10	0000..01	0011..01	0000..01	0001..10
K_B	11..0011	00..0010	11..0010	00..0001	11..0011	00..0010	11..0010	00..0001	01..0101

Table 11.7: Round-by-round rotational probabilities for Skein-512

Rounds	1-2	3	4-5	6-7	8-9	10-11	12-13	14-15
Prob. log_2	–	−6.7	−26.35	−17.05	−26.21	−17.05	−24.26	−17.05
Rounds	16-17	18-19	20-21	22-23	24-25	26-27	28-29	30-31
Prob. log_2	−28.26	−17.05	−28.29	−17.05	−23.79	−17.05	−23.56	−17.05
Rounds	32-33	34-35	36-37	38-39	40-41	42-43	44-45	46
Prob. log_2	−27.18	−17.05	−32.23	−17.05	−35.17	−17.05	−31.86	−6.7

11.2 Rotational Cryptanalysis of BMW-512

In this section we present rotational distinguishers for the compression functions of the submitted to the SHA-3 competition [105] BMW-512 [59], further denoted as BMWv1, and for the second, tweaked by the designers, version of BMW-512 [60], denoted as BMWv2. Thomsen in [134] described pseudo-collision and pseudo-preimage attacks on BMWv1. Practical differential distinguishers on BMWv2 were presented in [5, 63]. Our attack on BMWv1 is on the original design, while the attack on BMWv2 is on a modified version of the compression function, where one byte of the constant used in the internal function f_1 has been altered.

11.2.1 Analysis of BMWv1-512

BMWv1 takes two 1024-bit inputs: the message M and the chaining value H and produces an 1024-bit output. The compression function is constructed using three functions f_0, f_1 and f_2. Next, we give a short description of each function f_i. A complete specification of the functions can be found in [59]. All words in BMW-512 are 64 bits long. Assume that the message is $M = (M_0, \ldots, M_{15})$ and the chaining value is $H = (H_0, \ldots, H_{15})$.

The function f_0 takes as its input the pair: message M and chaining value H and produces an output $(Q_0, \ldots, Q_{15})$ as follows.

1. First intermediate words $W_0, \ldots, W_{15}$ are obtained as a bijective transformation of $M \oplus H$ defined below as

$$W_j = (M_{j_1} \oplus H_{j_1}) * (M_{j_2} \oplus H_{j_2}) * (M_{j_3} \oplus H_{j_3}) * (M_{j_4} \oplus H_{j_4}) * (M_{j_5} \oplus H_{j_5}),$$

   where $* \in \{+, -\}$, $j = 0, \ldots, 15$ and $j_1, j_2, \ldots, j_5 \in \{0, \ldots, 15\}$.

2. The words W_i undergo a bijective transformation and the output of f_0 are produced, i.e. $Q_j = s_j(W_j)$, where $s_j(x), j = 0, \ldots, 15$, are XORs of shifts and rotations of x (see Table 11.11).

The function f_1 takes as its input the pair: message M and output of f_0, and produces words $(Q_{16},\ldots,Q_{31})$ on its output. For $j=16,17$, Q_j are defined as:

$$Q_j = expand_1(j) = s_1(Q_{j-16}) + \ldots + s_0(Q_{j-1}) + AddElement(j-16),$$

while for $j=18,\ldots,31$, they are defined as:

$$Q_j = expand_2(j) = Q_{j-16} + \ldots + s_5(Q_{j-1}) + AddElement(j-16),$$

where $s_k(x)$ are the same functions as in f_0, and $AddElement(j) = M_j + M_{j+3} - M_{j+10} + K_{j+16}$. The constants K_j are obtained from the initial constant $C = 0x0555555555555555$ by multiplication, i.e. $K_j = j \cdot C$.

The last function f_2 produces 16 words of the new chaining value. It takes as its input the message M and $(Q_0,\ldots,Q_{31})$ (the outputs of f_0, f_1). First, it produces the words $XL = Q_{16} \oplus \ldots \oplus Q_{23}$ and $XH = XL \oplus Q_{24} \oplus \ldots \oplus Q_{31}$. Then, the first 8 words (out of 16) of the new chaining value are defined as[3]:

$$H_j = (SHL^{i_j}(XH) \oplus SHR^{k_j}(Q_{j+16}) \oplus M_j) + (XL \oplus Q_{j+24} \oplus Q_j),$$

where $j=0,\ldots,7$ and SHL^k, SHR^k are shifts to left and right by k bits.

We will build a rotational distinguisher for BMWv1 such that the input pairs of chaining values and message words will compose a rotational pair, but with some corrections. The output pairs of f_0 and f_1 will be rotational for all 1024 bits, while the output pairs of f_2 will be rotational for at least 384 bits.

11.2.1.1 Analysis of f_1, f_0, f_2

We start from f_1 because that is the only function that applies additions of constants. Note that in general, we cannot create a rotational pair for constants since their values are fixed. To overcome this technical difficulty, either constants have to be rotational, i.e. $K_j = K_j \lll_r$, or the errors from the constants have to be canceled with some other errors. In our attack, we will use the fact that the constants are almost rotational, and we will use small errors, coming from other words, to make the outputs fully rotational. Recall that the outputs $Q_j, j=16,\ldots,31$ of f_1 are defined as[4] $Q_j = AddElement(j-16) + s_1(Q_{j-16}) + \ldots + s_0(Q_{j-1})$. Let $T_j = s_1(Q_{j-16}) + \ldots + s_0(Q_{j-1})$ and $\tilde{T}_j = s_1(Q_{j-16} \lll_r) + \ldots + s_0(Q_{j-1} \lll_r)$. To obtain rotational outputs Q_j, we have to find an input message pair $(M, \tilde{M})$) for the following system of 16 equations:

$$[M_j + M_{j+3} - M_{j+10} + K_{j+16} + T_{j+16}] \lll_r = \tilde{M}_j + \tilde{M}_{j+3} - \tilde{M}_{j+10} + K_{j+16} + \tilde{T}_{j+16}, \quad (11.9)$$

$j=0,\ldots,15$. If we take into account the distributive properties of addition and rotation, then with some probability (that will be estimated later) this system can be rewritten as:

$$M_j \lll_r + M_{j+3} \lll_r - M_{j+10} \lll_r + K_{j+16} \lll_r + T_{j+16} \lll_r = \tilde{M}_j + \tilde{M}_{j+3} - \tilde{M}_{j+10} + K_{j+16} + \tilde{T}_{j+16}, \quad (11.10)$$

If we denote $M_j^{'} = M_j \lll_r - \tilde{M}_j$, then we obtain the following system:

$$M_j^{'} + M_{j+3}^{'} - M_{j+10}^{'} = K_{j+16} - K_{j+16} \lll_r + \tilde{T}_{j+16} - T_{j+16} \lll_r, \quad (11.11)$$

for $j=0,\ldots,15$. When the amount of rotation is 2, then the words $K_{j+16} - K_{j+16} \lll_2$, $j=0,\ldots,15$ have zeroes in all bytes except for the first and the last (the exact values

Table 11.8: Rotational properties of the constants of f_1 in BMWv1 and BMWv2

i	K_i	$K_i \lll_2$	$K_i - K_i \lll_2$
BMWv1			
16	5555555555555550	5555555555555541	000000000000000f
17	5aaaaaaaaaaaaaa5	6aaaaaaaaaaaaa95	f000000000000010
18	5fffffffffffffa	7fffffffffffffe9	e000000000000011
19	655555555555554f	955555555555553d	d000000000000012
20	6aaaaaaaaaaaaaa4	aaaaaaaaaaaaaa91	c000000000000013
21	6fffffffffffff9	bfffffffffffffe5	b000000000000014
22	755555555555554e	d555555555555539	a000000000000015
23	7aaaaaaaaaaaaaa3	eaaaaaaaaaaaaa8d	9000000000000016
24	7fffffffffffff8	ffffffffffffffe1	8000000000000017
25	855555555555554d	1555555555555536	7000000000000017
26	8aaaaaaaaaaaaaa2	2aaaaaaaaaaaaa8a	6000000000000018
27	8fffffffffffff7	3fffffffffffffde	5000000000000019
28	955555555555554c	5555555555555532	400000000000001a
29	9aaaaaaaaaaaaaa1	6aaaaaaaaaaaaa86	300000000000001b
30	9fffffffffffff6	7fffffffffffffda	200000000000001c
31	a55555555555554b	955555555555552e	100000000000001d
BMWv2			
16	5555555555555550	5555555555555541	000000000000000f
17	aaaaaaaaaaaaaaa5	aaaaaaaaaaaaaa96	000000000000000f
18	fffffffffffffffa	fffffffffffffffeb	000000000000000f
19	555555555555554f	555555555555553d	0000000000000012
20	aaaaaaaaaaaaaaa4	aaaaaaaaaaaaaa92	0000000000000012
21	fffffffffffffff9	fffffffffffffffe7	0000000000000012
22	555555555555554e	5555555555555539	0000000000000015
23	aaaaaaaaaaaaaaa3	aaaaaaaaaaaaaa8e	0000000000000015
24	fffffffffffffff8	fffffffffffffffe3	0000000000000015
25	555555555555554d	5555555555555535	0000000000000018
26	aaaaaaaaaaaaaaa2	aaaaaaaaaaaaaa8a	0000000000000018
27	fffffffffffffff7	fffffffffffffffdf	0000000000000018
28	555555555555554c	5555555555555531	000000000000001b
29	aaaaaaaaaaaaaaa1	aaaaaaaaaaaaaa86	000000000000001b
30	fffffffffffffff6	fffffffffffffffdb	000000000000001b
31	555555555555554b	555555555555552d	000000000000001e

are given in Table 11.8). On the other hand, for random $Q_j, j = 0,\ldots,15$, the difference $\tilde{T}_{j+16} - T_{j+16} \lll_2$ takes the values $0x16, 0x17$ with probability $2^{-8.4}$ when $expand_1(j)$ is applied, and $2^{-5.6}$ when $expand_2(j)$ is applied (this result is obtained experimentally, with 2^{27} trials). Hence, we can assume that the constant terms of System (11.11), have only two non-zero bytes – the first (MSB) and the last (LSB). For specific values of these terms (different values can be obtained since $\tilde{T}_{j+16} - T_{j+16} \lll_2$ takes two values, and there are 16 equations, therefore one can get 2^{16} systems), the words $M_j^{'}$ of the solution also have only two non-zero bytes, i.e. $M_j^{'}$ can be represented as $M_j^{'} = msb_j \cdot 2^{56} + lsb_j$, where $msb_j, lsb_j < 256$. The exact values of these bytes are given in Table 11.9. Once we have $M_j^{'}$, we can

Table 11.9: Constant terms and solutions for the systems in f_1 of BMWv1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\tilde{T}_{j+16} - T_{j+16} \lll_2$	16	16	16	16	16	16	16	16	17	16	16	16	16	17	16	17
LSB of R_j[a]	25	26	27	28	29	2a	2b	2c	2e	2d	2e	2f	30	32	32	34
MSB of R_j[a]	00	0f	0e	0d	0c	0b	0a	09	08	07	06	05	04	03	02	01
msb_j	59	18	e7	76	c5	d4	93	52	21	d0	cf	ce	ad	fc	0b	ea
lsb_j	28	28	29	29	30	32	31	28	25	22	2c	32	34	32	2f	2d

[a] $R_j = K_{j+16} - K_{j+16} \lll_2 + \tilde{T}_{j+16} - T_{j+16} \lll_2$

find the message pair. We choose the message words $M_j, \tilde{M}_j$ such that $\tilde{M}_j = M_j \lll_2 \oplus\delta_j$ (rotational with corrections δ_j). Since all $M_j^{'}$ were fixed by the system, we get the following equations:

$$M_j \lll_2 - M_j \lll_2 \oplus\delta_j = msb_j \cdot 2^{56} + lsb_j, j = 0,\ldots,15. \tag{11.12}$$

We would like to find many solutions (that we will use later) for this system. To do that, we fix the MSB of $M_j \lll_2$ and δ_j to msb_j and the LSB to lsb_j, i.e. $(M_j \lll_2)_{MSB} = (\delta_j)_{MSB} = msb_j, (M_j \lll_2)_{LSB} = (\delta_j)_{LSB} = lsb_j$. If we fix the middle 6 bytes of δ_j to 0, then the message words $M_j \lll_2 = msb_j \cdot 2^{56} + X_j \cdot 2^8 + lsb_j$, where $X_j < 2^{48}$, are solutions of (11.12). It is important to notice, that δ_j have only two non-zero bytes and therefore the input pairs of message words are rotational for 6 bytes. Hence, we can easily find $2^{16 \cdot 6 \cdot 8} = 2^{768}$ input rotational pairs of messages such that if the inputs $Q_0,\ldots,Q_{15}$ of f_1 are rotational, then the outputs $Q_{16},\ldots,Q_{31}$ are rotational as well.

Let us estimate the total probability of obtaining these rotational outputs. First, let us find the probability that System (11.9) is equivalent to System (11.10). For one equation the probability (obtained experimentally) is $2^{-3.8}$ and hence for the whole system it is 2^{-61}. Now, let us concentrate on the transformations in f_1. Since there are 2 applications of $expand_1(j)$ and 14 applications of $expand_2(j)$, the probability of obtaining the required differences $\tilde{T}_{j+16} - T_{j+16} \lll_2$ for all 16 outputs is $2^{-2 \cdot 8.4 - 14 \cdot 5.6} = 2^{-95.2}$. Therefore, the total rotational probability (obtained heuristically) of f_1 is $2^{-61-95.2} = 2^{-156.2}$.

The function f_0 uses the words $(M_i \oplus H_i)$ as inputs. Since the message pair is $(M_i, M_i \lll_2 \oplus\delta_i)$, instead of taking simply rotational inputs for the chaining values H_i, we will also introduce corrections. To obtain a fully rotational input for f_0 we will take the chaining value pairs $(H_i, H_i \lll_2 \oplus\delta_i)$. Then the input pair for f_0 is $(M_i \oplus H_i, M_i \lll_2 \oplus\delta_i \oplus H_i \lll_2 \oplus\delta_i) = (M_i \oplus H_i, (M_i \oplus H_i) \lll_2)$, hence it is rotational. Now let us find the probability that the outputs $Q_0,\ldots,Q_{15}$ are also rotational. These words are produced in two phases:

[3] We will use only these 8 words in our attack. Therefore we omit the definition of the next 8 words.

[4] The case when $expand_2(j)$ is used can be analyzed similarly.

1. the words $W_0, \ldots, W_{15}$ are generated as linear combinations of five terms of a type $M_i \oplus H_i$ or $-(M_i \oplus H_i)$,
2. each Q_i is obtained from W_i as $Q_i = s_i(W_i)$.

The rotational probabilities of the words W_i are given in Table 11.10.

Table 11.10: Rotational probabilities of the words in f_0 of BMWv1 and BMWv2

	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7
log_2	-3.82	-1.68	-3.82	-1.68	-1.68	-1.68	-3.82	-10.01
	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}	W_{14}	W_{15}
log_2	-3.82	-1.68	-3.82	-3.82	-1.68	-10.01	-3.82	-3.82

The theoretical basis for these numbers is provided by the Lemmas 11 ,12. Note that since we consider rotation amount $r = 2$ and the number of additions and substractions in W_i is limited to 4, the counter j of the sums in the lemmas takes only the value 0, hence the formulas for the probabilities can be significantly simplified for these specific values. The total probability of the first phase is $2^{-60.8}$. To compute the probability of the phase 2 of f_0 we only have to find the rotational probabilities of the functions s_i (see Table 11.11). There

Table 11.11: Rotational probabilities of the functions s_i used in BMW

Function	Definition	Prob. log_2
$s_0(x)$	$SHR^1(x) \oplus SHL^3(x) \oplus ROTL^4(x) \oplus ROTL^{19}(x)$	-4
$s_1(x)$	$SHR^1(x) \oplus SHL^2(x) \oplus ROTL^8(x) \oplus ROTL^{23}(x)$	-4
$s_2(x)$	$SHR^2(x) \oplus SHL^1(x) \oplus ROTL^{12}(x) \oplus ROTL^{25}(x)$	-4
$s_3(x)$	$SHR^2(x) \oplus SHL^2(x) \oplus ROTL^{15}(x) \oplus ROTL^{29}(x)$	-4
$s_4(x)$	$SHR^1(x) \oplus x$	-2
$s_5(x)$	$SHR^2(x) \oplus x$	-4

are 4, 3, 3, 3, 3 applications of s_0, s_1, s_2, s_3, s_4 respectively. Therefore, the probability of phase 2 is $2^{-4\cdot4-3\cdot4-3\cdot4-3\cdot4-3\cdot2} = 2^{-58}$, and hence, the total rotational probability of f_0 is $2^{-118.8}$.

The function f_2 takes the message M and the words $Q_0, \ldots, Q_{31}$ (outputs of f_0, f_1) as input, and produces the next chaining value. We can assume the words $Q_0, \ldots, Q_{31}$ to be rotational with some probability (the combined rotational probabilities of f_0 and f_1). The terms XL and XH are rotational with probability 1 since they are produced as XORs of rotational words. We require rotational outputs from the shifts of XH and the shifts of $Q_j, j = 16, \ldots, 23$. The rotational probability of all the shifts of XH is $2^{-4-4-4-2-4-4-4-4} = 2^{-30}$, while for the shifts of Q_j we pay $2^{-4-4-4-4-4-4-4-4} = 2^{-28}$ (see Table 11.11). Since the message pair words are rotational in 6 bytes, it follows that the words $SHL^{j_k}(XH) \oplus SHR^{j_l}(Q_j) \oplus M_j$ are also rotational in 6 bytes (all bytes except MSB and LSB). Let $P_j = SHL^{j_k}(XH) \oplus SHR^{j_l}(Q_j) \oplus M_j$ and $R_j = XL \oplus Q_{j+24} \oplus Q_j$. Then the chaining values are defined as $H_j = P_j + R_j$. Note that P_j is rotational for 6 bytes, and R_j is fully rotational. For the error of the new chaining value we have $(P_j + R_j) \lll_2 -[(P_j \lll_2 \oplus\delta_j) + R_j \lll_2]^{2^{-1.68}} = P_j \lll_2 + R_j \lll_2 -(P_j \lll_2 \oplus\delta_j) - R_j \lll_2 = P_j \lll_2 -(P_j \lll_2 \oplus\delta_j)$, hence the error can be only in the MSB and LSB if there are no carries (with probability 2^{-1}) in the LSB. Therefore, for the rotational properties of the first 8 chaining values in 6 bytes we have to pay in total $2^{-8\cdot(1.68+1)}$. If we take into account the previous probabilities of the shifts, we get the total rotational probability of f_2, which is $2^{-30-28-21.5} = 2^{-79.5}$. The output pair is rotational in at least $8 \cdot 6 = 48$ bytes, or 384 bits.

11.2.1.2 The Attack on the Full BMWv1

The relations between the pairs of input message words and the chaining value words are fully fixed. For the first input, the message words M_j are chosen randomly, except their MSB and LSB which are fixed as explained above. The chaining values H_j are chosen randomly as well. Then, the message $\tilde{M}$ and the chaining value $\tilde{H}$ of the second input are defined as $\tilde{M}_j = M_j \lll_2 \oplus\delta_j, \tilde{H}_j = H_j \lll_2 \oplus\delta_j$. The probability that such input pair will produce output pair of chaining values, rotational in 384 bits, is the combined probability of f_0, f_1, f_2 which is $2^{-156.2-118.8-79.5} = 2^{-354.5}$. On the other hand, the same probability for a random function is 2^{-384}, hence BMWv1 can be distinguished from a random function.

Note, we can obtain non-random properties for the last 8 chaining values $H_8, \ldots, H_{15}$ as well[5]. We only have to take into account the terms $ROTL^{(j+1)}(H_k)$ which are rotational in 6 bytes. Also, the complexity of the whole attack can be reduced to $2^{223.5}$ compression function calls by using more advanced techniques.

11.2.2 Analysis of Modified Version of BMWv2-512

The compression function BMWv2 is similar to the one of BMWv1, but a few tweaks are introduced by the designers. We will describe only the differences between these two function. The first tweak is in f_0, where the words Q_j are produced as $Q_j = s_j(W_j) + H_{j+1}$. The second tweak is in f_1. Now this function takes the chaining value H as an additional input. The tweak of f_1 is in the AddElement function, which is defined as follows

$$AddElement(j) = (M_j \lll_{j+1} + M_{j+3} \lll_{j+4} - M_{j+10} \lll_{j+11} + K_{j+16}) \oplus H_{j+7}.$$

We attack a modified version of BMWv2, denoted as BMWv2$_C$, where the above round constants K_{j+16} are obtained by multiplying the round indexes $(j+16)$ by the constant $C = 0x5555555555555555$. In the original version the value of the constant is $C = 0x0555555555555555$.

11.2.2.1 The Attack on BMWv2$_C$

We will take different approach for producing rotational pairs in BMWv2$_C$ although the analysis uses the results of the previous section. The input pairs of messages and chaining values will be fully rotational, while the output chaining values will be rotational in the first 8 words (512 bits).

Let us fix random a message $(M_0, \ldots, M_{15})$ and a chaining value $(H_0, \ldots, H_{15})$ for the first input of f_0, and the pair $(M_0 \lll_2, \ldots, M_{15} \lll_2)$, $(H_0 \lll_2, \ldots, H_{15} \lll_2)$ for the second. Since f_0 in BMWv2 differs from f_0 in BMWv1 only in the extra additions of H_j in BMWv2, in order to find the rotational probability of f_0, we only have to consider these 16 additions. Thus, the probability of rotational output pair for f_0 is $2^{-118.8-16\cdot 1.68} = 2^{-145.7}$.

Now, let us focus on f_1. When the constant C is fixed to $0x5555555555555555$ then the values of the differences $K_{j+16} - K_{j+16} \lll_2$ are only one byte (see Table 11.8).

On the other hand, all of these differences (rotational errors of the constants) can be canceled since addition and rotation are not fully distributive. For example, for some x, y the following holds $(x+y) \lll_2 = x \lll_2 + y \lll_2 + 1$. When more terms are added, these errors can be larger, i.e. for some $x_1, \ldots, x_k$ it holds $(x_1 + \ldots + x_k) \lll_2 = x_1 \lll_2 + \ldots + x_k \lll_2 + e_+$, where $e_+ \in \{1, 2, \ldots\}$. We have found that all differences $K_{j+16} - K_{j+16} \lll_2$ can be canceled with these errors coming from the additions/rotations. When the input pairs of words

[5] We omit the description since we already have an attack.

$Q_0, \ldots, Q_{15}, M_0, \ldots, M_{15}, H_0, \ldots, H_{15}$ are rotational, then the probabilities of rotational output pairs for $expand_1(j), j = 16, 17$ and $expand_2(j), j = 18 \ldots, 31$ (obtained experimentally) are given in the Table 11.12. The total probability of obtaining all 16 rotational outputs

Table 11.12: Rotational properties of the words in f_1 (without the shifts) in BMWv1

	Q_{16}	Q_{17}	Q_{18}	Q_{19}	Q_{20}	Q_{21}	Q_{22}	Q_{23}
log_2	-2.37	-2.38	-3.93	-3.95	-3.97	-3.97	-3.98	-4.00
	Q_{24}	Q_{25}	Q_{26}	Q_{27}	Q_{28}	Q_{29}	Q_{30}	Q_{31}
log_2	-4.00	-4.02	-4.03	-4.03	-4.03	-4.03	-4.03	-4.04

from these transformations, i.e. the rotational probability of f_1 is around 2^{-61}.

Finally, let us analyze f_2. We require rotational outputs only for the first 8 new chaining values, i.e. $H_0, \ldots, H_7$. Similarly as for BMWv1, the probability can be estimated simply by counting the number of shifts and additions required for producing these 8 values, i.e. the probability is $2^{-30-28-8 \cdot 1.68} = 2^{-71.5}$. Note that now there are no corrections in the message words, hence the 512-bit output is fully rotational.

For the whole BMWv2$_C$, the probability that rotational inputs of messages and chaining values will produce rotational outputs in the first 8 words is $2^{-145.7-61-71.5} = 2^{-278.2}$. On the other hand, a rotational input in a random function, will produce a rotational output in 8 words (512 bits) with probability 2^{-512}. The probability of our distinguishers most likely can be raised if message modification technique is applied. Then the first phase of f_0 (probability 2^{-61}) can be passed for free.

The low attack complexity allows us to launch rotational distinguishers for the 384-bit version of BMWv2$_C$ as well. Note that, increasing the number of applications of $expand_1(j)$ (which is considered to be stronger) from 2 to all 16 does not stop the attack because the rotational probability of $expand_1(j)$ is higher than the one of $expand_2(j)$. Also, the probability of rotational output pairs, does not seems to change significantly, when the order of s_j in f_0 and f_1 is changed.

Chapter 12

Meet-in-the-middle Attacks on SHA-3 Candidates

Since most of the attacks on hash functions have been differential-based collision attacks, the majority of the SHA-3 designs claimed resistance against differential cryptanalysis while less attention was given to the resistance against other attacks. The subject of this analysis is meet-in-the-middle attacks and their application to preimage search.

A meet-in-the-middle attack on a cryptographic primitive is applicable if the execution can be expressed as a sequence of transformations all of which have at least one input that is independent of the other transformations. Providing the invertibility of the last transformation, the full execution can be divided into independent parts, which are connected using the birthday paradox.

One of the first such attack was the attack on Double-DES [48]. Double-DES, being composed of two consecutive iterations of single DES with different keys, was found to be vulnerable to the following meet-in-the-middle attack: given a pair (*plaintext*, *ciphertext*) one can find a Double-DES key (a pair of single DES keys), which is valid for this pair, with complexity of about 2^{32} encryptions. A full attack on Double-DES, which gives the real key, is based on this approach as well and it is faster than the brute-force.

Meet-in-the-middle attacks on hash functions based on the Merkle-Damgård construction are hard to apply since the compression function is usually assumed to be non-invertible. The alternative sponge construction [15] allows invertible transformations, but requires the internal state to be large so that the meet-in-the-middle approach would have a complexity higher than the complexity of the simple brute force.

Surprisingly, several SHA-3 proposals are vulnerable to this type of attack. In this section we describe meet-in-the-middle based preimage attacks on Boole [122], Edon-R [61], and Sarmal [138]. Two ideas are common for all the attacks. First, all the functions have invertible (or partially invertible) transformations, which allows us to execute the meet-in-the-middle. Secondly, we reduce the intermediate state space exploiting the non-random behavior of the round transformations.

The analysis presented further is composed as follows. First, we describe the meet-in-the-middle preimage attack in general and remind how it can be maintained with little memory. Then we show how preimages for Boole, Edon-R, and Sarmal can be found. We also discuss possible computation-memory tradeoffs.

Table 12.1: Complexity of the preimage attacks described in this section.

	Computations	Memory
Boole-384/512	2^{288}	2^{64}
Edon-R-n	$2^{n-s} + 2^{n-k+s}$	$2^s + 2^k$
	$2^{n-s} + 2^{\frac{n}{2}+s+32.5}$	2^s
Sarmal-512	$2^{512-s} + 2^{256+s}$	2^s

12.1 Meet-in-the-Middle Attacks on Hash Functions

Hash functions with invertible compression functions become susceptible to preimage attacks if the size of the internal state is too small. Preimages can be obtained by performing a meet-in-the-middle attack on the compression function. In this section we will describe this generic scenario in more details.

Let $F : D \to D$ and $G : D \to D$ be two permutations and $H = G \circ F$ the composition of these permutations. In our setting, the function H is the hash function, F is defined as the compression function with a fixed IV and G is the inverse of the compression function for a fixed target value. Furthermore, we define auxiliary functions $\pi_{1,2} : D \times D \to D$ that map tuples to their first, respectively second component.

Assume we want to perform a meet-in-the-middle attack on h. The standard technique is to compute two sets

$$S_1 = \{(F(x), x) : x \in_R D\} \quad \text{and} \quad S_2 = \left\{\left(G^{-1}(y), y\right) : y \in_R D\right\}$$

such that $|S_1| \cdot |S_2| = |D|$. Either sorting these two sets in their first component or computing them in such a way that they are already ordered in this component allows us to easily find colliding values

$$\pi_1((F(x), x)) = \pi_1\left((G^{-1}(y), y)\right)$$

by comparing the elements of the two sets in linear time. Each collision gives us a pair (x, y) such that $H(x) = y$. How to balance the size of the sets S_1 and S_2 depends on the relative cost of the function G^{-1} compared to an evaluation of the function F. It may for instance be that G is easily invertible, meaning an evaluation of G^{-1} costs about the same number of operations as an evaluation of the function F. In this case we choose the sets S_1 and S_2 to be of equal size $\left\lceil \sqrt{|D|} \right\rceil$. However, if the evaluation of G^{-1} is k times more expensive than the evaluation of F, we should choose the set $|S_1|$ to be of size $\sqrt{k \cdot |D|}$ and S_2 of size $\sqrt{k^{-1} \cdot |D|}$ to obtain a minimum number of overall operations. The memory complexity of this naive approach is non-negligible however: We need to store a total of $2 \cdot \left(\sqrt{|D|}(\sqrt{k} + \sqrt{k^{-1}}\right)$ elements of the domain D to carry it out. Storing both sets is not really necessary: Only the smaller should be stored, the values of the larger can be computed on the fly and compared against the elements of the smaller set.

In some cases the memory requirement can be completely eliminated by a technique based on Floyd cycle finding first described in an article by Morita, Ohta and Miyaguchi

[101]. Although several works on hash functions refer to memoryless variants of meet-in-the-middle attacks [117, 96], all of them cite either one or both papers by Quisquater and Delescaille on collision search for DES [120, 119]. These two papers however do not directly deal with meet-in-the-middle attacks, but describe the technique of using distinguished points for collision search. Oorschot and Wiener describe the same technique for memoryless meet-in-the-middle later in [137].

12.1.1 Eliminating the memory requirement

Assume we are given another function $r : D \to \{0,1\}$ which maps elements of the domain D to a single bit in a random fashion. Using this *switching function* we can define a step function s that evaluates x either to $F(x)$ or to $G(x)$, depending on the value of x:

$$s : D \to D, \quad x \mapsto \begin{cases} F(x) & \text{if } r(x) = 0 \\ G(x) & \text{if } r(x) = 1 \end{cases}$$

This function s can then be used in a Floyd cycle finding algorithm: We start from a random value $x \in D$ and use just two elements $a = s(x)$ and $b = s^2(x)$. In each step we then update a by applying s to it and b by applying s^2 to it. Upon finding a cycle, we must check whether we really have found a pair $F(x) = G^{-1}(y)$ or whether we have found a cycle in F or in G. If the output of r is equidistributed, for each cycle we find $\Pr(F(x) = G^{-1}(y)) = 0.5$. In case of encountering a cycle in F or G we restart the algorithm with another random element $x \in D$.

Significant problems can arise if the output of r is not equidistributed, for instance if G is very costly to compute relative to F and we want to simulate the case of $|S_1| = k \cdot |S_2|$ with k large.

For the hash functions that we attack we define two functions F and G that are used in the memoryless approach. The F function is used for the forward direction and the G function is used for the backward one. The switching function r is defined as the parity of x.

12.1.2 Reduced state principle

The meet-in-the-middle (MITM) attack needs a collision in the intermediate state. However, the state may be so large that a straightforward application of the MITM approach would require more than 2^n computations for a n-bit hash digest. Thus the generic principle we use further is to generate intermediate states only from a smaller subspace (where some bits are fixed to zero) thus reducing the birthday dimension and the complexity of the attack.

The generic framework is defined as follows. A hash function with an n-bit digest has an internal state of size k bits. We manage to get intermediate states with t bits fixed to 0. Then to get a MITM connection we need to get two states that collide in $(k - t)$ bits so that the *birthday space* D has size 2^{k-t}. This implies that we must get two sets S_1 and S_2 such that $|S_1| \cdot |S_2| = 2^{k-t}$. The exact ratio between S_1 and S_2 is defined by the complexity of inverting the compression (round) function.

For the memoryless version of the MITM attack, we need to tweak the attack slightly such that we can define the functions F and G. Each of the functions is a composition of two functions, first projecting the birthday space into the state space, the second mapping the state space into the birthday space again (fixing some bits to zero). In other words, let $F = f \circ \mu$ and $G = g \circ \nu$. When memoryless meet-in-the-middle is possible in our attacks we will define these functions accordingly.

12.2 Boole

Boole is a family of hash functions [122] based on a stream design. Internally, Boole has a large state $\sigma_t = (R_t[0], R_t[1], \ldots, R_t[15])$ of 16 words plus 3 additional word accumulators denoted by l_t, x_t, and r_t (t is the time). The words are 64 bits each. Hashing a message in Boole is done in three phases: 1)Input phase, where the whole message is processed word by word, and for each input word the state and the accumulators are updated, 2)mixing phase, where only the state is updated depending on the values of the accumulators, 3)output phase, where the output is produced.

The *update of the state*, referred to as a *cycle*, is defined as:

$$\begin{aligned} R_{t+1}[i] \leftarrow &\quad R_t[i+1], \text{for } i = 1 \ldots 14 \\ R_{t+1}[15] \leftarrow &\quad f_1(R_t[12] \oplus R_t[13]) \oplus (R_t[0] \lll 1) \\ R_{t+1}[0] \leftarrow &\quad R_{t+1}[0] \oplus f_2(R_{t+1}[2] \oplus R_{t+1}[15]), \end{aligned}$$

where f_1 and f_2 are some non-linear functions, intended to simulate random functions.

Let w_t be a message word. The *update of the accumulators* is defined as:

$$\begin{aligned} temp \leftarrow &\quad f_1(l_t) \oplus w_t \\ l_{t+1} \leftarrow &\quad temp \lll 1 \\ x_{t+1} \leftarrow &\quad x_t \oplus w_t \\ r_{t+1} \leftarrow &\quad (r_t \oplus temp) \ggg 1 \end{aligned}$$

The whole message is absorbed in the input phase. Sequentially, for each message word w_t the following is done:

1. *update the accumulators*
2. $R_t[3] \leftarrow R_t[3] \oplus l_{t+1}$
3. $R_t[13] \leftarrow R_t[13] \oplus r_{t+1}$
4. *update the state (cycle)*

The mixing phase is invertible and its description is irrelevant in our attack.

Each iteration of the output phase produces one output word. One iteration is defined as:

1. *cycle*
2. Output the word $v = R[0] \oplus R[8] \oplus R[12]$

For example, the output for Boole-256 is produced in 8 iterations.

Let us present two observations about the invertibility of the update functions of the state and the accumulators.

Observation 3 *The state update (*cycle*) is an invertible function. If a new state σ_{t+1} is given, then the state σ_t that produced σ_{t+1} in a single* cycle *can be found from the following equations:*

$$\begin{aligned} R_t[0] = &\quad (R_{t+1}[15] \oplus f_1(R_{t+1}[11] \oplus R_{t+1}[12])) \ggg 1 \\ R_t[1] = &\quad R_{t+1}[0] \oplus f_2(R_{t+1}[2] \oplus R_{t+1}[15]) \\ R_t[i] = &\quad R_{t+1}[i-1], i = 2, \ldots 15 \end{aligned}$$

Observation 4 *The update of the accumulators can be inverted with probability* $1 - 1/e$. *If the values of the new accumulators* $l_{t+1}, x_{t+1}, r_{t+1}$ *and the input message word* w_t *are fixed, the values of the previous accumulators* l_t, x_t, r_t *are determined as:*

$$\begin{aligned} l_t &= f_1^{-1}((l_{t+1} \ggg 1) \oplus w_t) \\ x_t &= x_{t+1} \oplus w_t \\ r_t &= r_{t+1} \lll 1 \oplus f_1(l_t \oplus w_t) \end{aligned}$$

Moreover, if the values of l_t, l_{t+1} *(or* r_t, r_{t+1}*) are fixed, the value of the message word* w_t *can be found uniquely:*

$$\begin{aligned} w_t &= (l_t + 1 \ggg 1) \oplus f_1(l_t) \\ (\quad w_t &= (r_{t+1} \lll 1) \oplus r_t \oplus f_1(l_t)) \end{aligned}$$

In order to invert the function f_1 we will use a look-up table $(x, f_1(x))$ with all 2^{64} values for x, sorted by the second entry. Then, a inversion of $f_1(x)$ is equivalent to a look-up in this table.

12.2.1 Preimage attack on Boole-384 and Boole-512

The intermediate state of Boole has 16 state words and 3 accumulators, hence 19 words in total. Further, we will show how to fix the values of the state words $R[3], \ldots, R[12]$ (10 words in total) to zero in forward and backward directions. This will mean that $k = 19 \cdot 64 = 1216$ and $t = 10 \cdot 64 = 640$, and the birthday space D has only 9 words (576 bits). We will also define $f(x)$ and $g(x)$ for the memoryless MITM attack.

Defining μ - fixing $R[3], R[4], \ldots, R[12]$ forwards. From the description of the input phase it follows that:

$$R_{10}[3] = R_9[4] = \ldots = R_1[12] = R_0[13] \oplus r_1$$

Note that the value of r_1 can be controlled with w_0 (Observation 3). Hence, if we take $r_1 = R_0[13]$, we will get $R_{10}[3] = 0$. Similarly, for $R_{10}[4]$ we have:

$$R_{10}[4] = R_9[5] = \ldots = R_2[12] = R_1[13] \oplus r_2$$

We can change the value of r_2 with w_1 such that $R_1[13] \oplus r_2 = 0$ holds. Then $R_{10}[4] = 0$. The same technique can be applied for fixing the values of $R_{10}[5], \ldots, R_{10}[12]$.

Note that we can not fix the values of more than these 10 words. When we control the value of r_t with the input word w_{t-1}, it means that we also change the value of l_t (which is added to $R_t[3]$). Since we can not control the value of both accumulators with a single message word, and both of them are xor-ed into the registers $R[3]$ and $R[13]$, it means that we can not control the values of more than 10 words.

Defining $f(y)$ for the memoryless MITM attack. The birthday space D has 9 words. Let $y = y_1 || y_2 || \ldots || y_9$, then $f(y)$ can be defined as compression of the input words y_i with $1 \leq i \leq 9$ in the first 9 *cycles*. Thus when fixing $R[3], \ldots, R[12]$ in forward direction, we first compress y, and then we start with our technique for fixing these words to zero (function μ).

Defining v - fixing $R[3], R[4], \ldots, R[12]$ backwards. Our backwards strategy is the following: first we invert the output and the mixing phase and obtain one valid intermediate state. Then, by changing the input words, we fix $R[3], R[4], \ldots, R[12]$.

First, let us deal with the inversion of the output phase. In each *cycle* of this phase one output word is produced. Hence, the digest is produced in 8 cycles[1]. The output word v_t is defined as $v_t = R_t[0] \oplus R_t[8] \oplus R_t[12]$. Let $H^* = (h_0, \ldots, h_7)$ be the target hash value. We have to construct a state $\sigma_t = (R_t[0], \ldots, R_t[15])$ such that $h_0 = v_t, h_1 = v_{t+1}, \ldots, h_7 = v_{t+7}$. First, we put any values in $R_t[0], R_t[9], R_t[10], \ldots, R_t[15]$. The rest of the words are undefined. Then, we find $R_t[8]$ from the equation $R_t[8] = R_t[0] \oplus R_t[12] \oplus h_0$. Obviously we get that $v_t = h_0$. After the *cycle* update we obtain a new state σ_{t+1}. Then, we determine the value of $R_t[1]$ from the equation $R_t[1] = R_{t+1}[0] = R_{t+1}[8] \oplus R_{t+1}[12] \oplus h_1$, and therefore $h_1 = v_{t+1}$. The values for $R_t[2], \ldots, R_t[7]$ are determined similarly. This way we can define the rest of the words in the state σ_t, which in the 7 sequential *cycle* updates produces the target hash value.

Let us fix the accumulators to any values. Then, inverting the mixing phase is trivial because the length of the preimage, as shown further, is known and the values of the accumulators are also known.

Now that we have inverted the output and mixing phase, we have the freedom of choosing the input message words. The technique for fixing is rather similar to the one used for fixing this set in forward direction. But in the backward direction, we control the values of the l_t accumulators (rather then the values of r_t as in the forward direction) with the input words w_t (Observation 4). From the description of the input phase we get:

$$R_{10}[12] = R_{11}[11] = \ldots = R_{18}[4] = R_{19}[3] \oplus l_{20}$$

Therefore if we take $l_{20} = R_{19}[3]$ we will get $R_{10}[12] = 0$. Similarly, for $R_{10}[11]$ we have:

$$R_{10}[11] = R_{11}[10] = \ldots = R_{17}[4] = R_{18}[3] \oplus l_{19}$$

If we take $l_{19} = R_{18}[3]$ we obtain $R_{10}[11] = 0$. The same technique can be used to fix the variables $R_{10}[10], \ldots, R_{10}[3]$.

One may argue that for controlling the values of the l_t registers when going backwards we have to pay an additional cost because f_1 is not always invertible. But we have to keep in mind that there are values for which f_1 has many inversions. Hence, if we start with a set of N different values, we can expect to find N different inversions for these values and thus we do not have to repeat the inversion.

Defining $g(y)$ for the memoryless MITM attack. The function $g(y)$, where $y = y_1||y_2||\ldots||y_9$, is defined as 9 consecutive backward rounds of the input phase with inputs y_i. The starting state of these 9 rounds is the state obtained after the inversion of the output and mixing phases (as described above). Note that after the application of the function $g(y)$ a new state is obtained. Then, to this state, we apply our technique for fixing $R[3], \ldots, R[12]$ in 10 backwards rounds (function v).

12.2.2 Complexity of the Attack

The preimage that we obtained has a length of at least $9 + 9 + 10 + 10 = 38$ words. The memoryless MITM attack requires about $2^{\frac{9 \cdot 64}{2}} = 2^{288}$ computations[2] and 2^{64} memory (for inverting f_1).

[1] In Boole-384, the output is produced in 6 cycles.

[2] One computation is equivalent to one round of the input phase or one round of the mixing phase.

12.3 Edon-R

The hash family Edon-R [61] uses the well known Merkle-Damgård design principle. The intermediate hash value is rather large, two times the digest length[3]. For an n-bit digest the chaining value H_i of Edon-R is composed of two block of n bits each, i.e. $H_i = (H_i^1, H_i^2)$. The message input M_i for the compression function is also composed of two blocks, i.e. $M_i = (M_i^1, M_i^2)$. Let Edon be the compression function. Then the new chaining value is produced as follows:

$$H_{i+1} = (H_{i+1}^1, H_{i+1}^2) = \text{Edon}(M_i^1, M_i^2, H_i^1, H_i^2)$$

The hash value of a message is the value of second block of the last chaining value.

Internally, the state of Edon-R has two n-bit blocks, A and B. The compression function of Edon-R consists of eight updates, each being an application of the quasigroup operation $Q(x, y)$[4], to one of these blocks. With A_i and B_i we will denote the values of these blocks after the i-th update in the compression function (please refer to Fig. 12.1). Hence, each input pair (H_i, M_i) generates internal state blocks $(A_1, B_1), (A_2, B_2), \ldots, (A_8, B_8)$. The new chaining value (the output of the compression function) H_{i+1} is the value of the blocks (A_8, B_8).

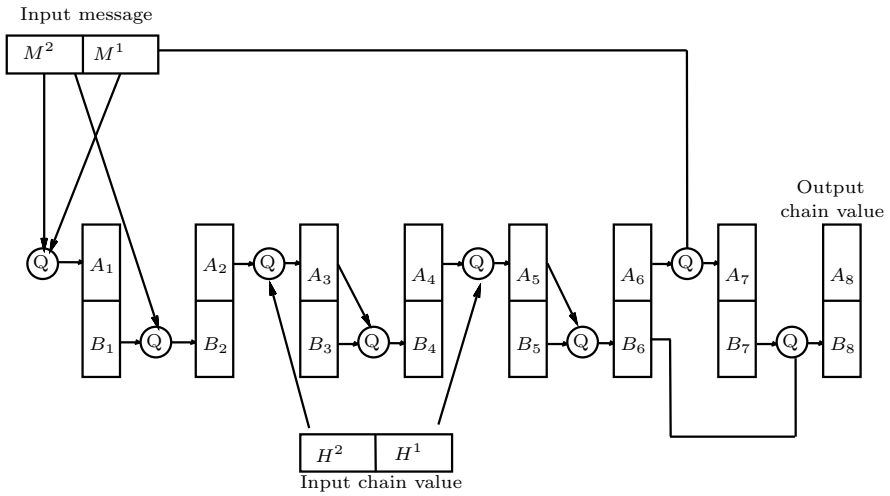

Figure 12.1: Outline of Edon-R compression function

Let us present a simple observation that is used in the attack.

Observation 5 *The quasigroup operation $Q(x, y)$ of Edon-R is easily invertible, i.e. if A and C (B and C) are fixed then one can easily find B (C) such that $Q(A, B) = C$.*

[3]Edon-224 and Edon-384 have 512 and 1024 bits chaining values, respectively.
[4]The exact definition of the quasigroup operation can be found in [61].

12.3.1 Preimage attack on EDON-R-n

The internal state of EDON-R-n (the chaining value $H = H_1||H_2$) has $2n$ bits. We will show how to fix $H_1 = 0$. Then the preimage attack can be mounted using the MITM approach (Section 12.1), where $k = 2n$ and $t = n$. The backward step is time-consuming so we will use the memory MITM attack.

Fixing H_1 in forward direction. We need only one message block to get the desired $H_1^{\text{new}} = 0$. Both initial value blocks are fixed as well. We claim that for each M_1 we can find M_2 such that this message input and the initial value blocks will produced a zero value in H_1^{new}.

Indeed, let M_1 be set to some random value. Then we obtain the value of A_6 since $A_7 = H_1^{\text{new}} = 0$ and the function Q is invertible. We consecutively obtain the values of A_5,A_4,A_3,A_2, and A_1 (keep in mind that the initial chaining value is fixed). Given A_1 and M_1, we derive M_2 by inverting the first application of Q. Finally we obtain all B's and thus a pair $(H_1^{\text{new}} = 0, H_2^{\text{new}})$.

Fixing H_1 in backward direction. We need only one step (one message block) to get a pair of the form $(0, H_2)$ from a given hash value $H = H_2^{\text{new}}$.

First, we set M_1 to some predefined value m. Then we assign to A_8 some random value and consecutively obtain the values of the following internal variables (in this order): A_7, B_7, B_6, A_6 (using M_1), A_5, B_5, B_4, A_4, A_3, B_3. We repeat this step 2^k times for different values of A_8 and store 2^k different pairs (A_3, B_3).

Now we set M_2 to some random value[5] and obtain the values of A_1, A_2, and B_2 using the value of M_1. If we repeat this step 2^{n-k+s} times then we will find 2^s different values of B_2 that coincide with some values of B_3 from the stored set. For each of these values we define H_2 such that $Q(A_2, H_2) = A_3$. The complexity requirements for this part are: 2^{n-k+s} computations[6], where $s - k < 65$, and $2^s + 2^k$ memory.

These 2^s pairs can be obtained using the memoryless MITM as well, where the MITM space is the value of B_2. Because of the message padding we should take any $n - 65$ bits of B_2 so that the input and the output of the MITM function F and G would have the same size. The $(n - 65)$-bit input to the function F is padded with the message padding, and the input to the function G is padded, for example, with zeros. Then, if a $(n - 65)$-bit collision between F and G is obtained, the probability that they coincide in the rest of the 65 bits is 2^{-65}. Hence, for constructing 2^s pseudo preimages with the memoryless MITM, one needs $2^s \cdot 2^{\frac{n-65}{2}+65} = 2^{\frac{n}{2}+s+32.5}$.

12.3.2 Complexity of the Attack

Starting from the initial value, we generate 2^{n-s} different chaining values with $H_1 = 0$. Note that we do not store these values. Then, with high probability, we can expect that one of these values will be in the set of the 2^s pseudo preimages generated in the backward direction. Under the condition $s - k < 65$ the total complexity of the attack when memory is used in the backward step is $2^{n-s} + 2^{n-k+s}$ computations and $2^s + 2^k$ memory. If only negligible memory in the backward step is used the computational complexity is $2^{n-s} + 2^{\frac{n}{2}+s+32.5}$ at the same time needing 2^s memory.

[5] The value is not truly random: 65 bits of the last message block are reserved for padding.

[6] Here and below, one computation is not more than one compression function call.

12.4 Sarmal

Sarmal-n [138] is a hash family based on the HAIFA design. After the standard padding procedure, the padded message is divided into blocks of 1024 bits each, i.e. $M = M_1||M_2||\ldots||M_k, |M_i| = 1024, i = 1,\ldots,k$. Each block is processed by the compression functions. HAIFA design implies that the compression function f has four input arguments: the previous chain value h_{i-1}, the message block M_i, the salt s, and the block index t_i. Hence, h_i is defined as $h_i = f(h_{i-1}, M_i, s, t_i)$. The final chaining value h_k is the hash value of the whole message M. For Sarmal-n the chaining value h_i has 512 bits. Let us denote the left and the right half of h_i as L_i and R_i respectively, i.e. $h_i = L_i||R_i$. The salt s has 256 bits (similarly let $s = s_1||s_2$), and the block index t_i has 64 bits. Then, the compression function of Sarmal-n can be defined as:

$$f(h_{i-1}, M_i, s, t_i) = \mu(L_{i-1}||s_l||c_1||t_i, M_i) \oplus \nu(R_{i-1}||s_r||c_2||t_i, M_i) \oplus h_{i-1}, \tag{12.1}$$

where μ and ν are functions that output 512 bit values, and c_1, c_2 are some constants. The exact definition of these functions is irrelevant for our attack.

12.4.1 Preimage Attack on Sarmal-512

We will show how to invert the compression function of Sarmal-512. Note that the intermediate chaining value of Sarmal has 512 bits. Then the preimage attack can be launched using the MITM approach (Section 12.1), where $k = 512$ and $t = 0$. The inversion of the compression function is time-consuming so we will use the memory MITM attack.

Going forward from the IV. Since we do not fix anything ($t = 0$), going forward from the IV is trivial. We simply generate a number of intermediate chaining values, by taking different random messages as an input for the first compression function.

Going backward from the target hash value. Let us explain how the compression function can be inverted.

From (12.1) we get:

$$\begin{aligned}
f \quad (\quad & h_{i-1}, M_i, s, t_i) = \\
= \quad \mu \quad & (L_{i-1}||s_l||c_1||t_i, M_i) \oplus \nu(R_{i-1}||s_r||c_2||t_i, M_i) \oplus h_{i-1} = \\
= \quad \mu \quad & (L_{i-1}||s_l||c_1||t_i, M_i) \oplus \nu(R_{i-1}||s_r||c_2||t_i, M_i) \oplus L_{i-1}||R_{i-1} = \\
= \quad \mu \quad & (L_{i-1}||s_l||c_1||t_i, M_i) \oplus \nu(R_{i-1}||s_r||c_2||t_i, M_i) \oplus L_{i-1}||0 \oplus 0||R_{i-1} = \\
= \quad (\mu \quad & (L_{i-1}||s_l||c_1||t_i, M_i) \oplus L_{i-1}||0) \oplus (\nu(R_{i-1}||s_r||c_2||t_i, M_i) \oplus 0||R_{i-1})
\end{aligned}$$

Let us fix the values of M_i, s, and t_i. Then, we can introduce the functions $F(L_{i-1}) = \mu(L_{i-1}||s_l||c_1||t_i) \oplus L_{i-1}||0$, and $G(R_{i-1}) = \nu(R_{i-1}||s_r||c_2||t_i) \oplus 0||R_{i-1}$. Let H^* be the target hash value. Then we get the equation:

$$F(L) \oplus G(R) = H^*$$

If we generate 2^{256} different values for $F(L)$ and the same amount for $G(R)$, then, by the birthday paradox, with high probability we can expect to get at least one pair $(F(L_l), G(R_m)$ that will satisfy the above equation and therefore obtain that $h = L_l||R_m$ is a preimage of H^*.

A memoryless version of this pseudo-preimage attack can be obtained by introducing the function $\tilde{F}(L) = F(L) \oplus H^*$, and launching the memoryless MITM attack on $\tilde{F}$ and G. This would require 2^{256} computations and negligible memory.

12.4.2 Complexity of the Attack

Since the backward direction, i.e. inverting the compression function, is time consuming we will use the memory version of MITM attack. Going backwards from the target hash value we create a set S_2 of 2^s different chaining values. To create this set we need $2^{256} \cdot 2^s = 2^{256+s}$ computations. Then, starting from the initial value, we generate 2^{512-s} different chaining values. Note, we do not store these values, we store only the smaller set S_2. Then, with a high probability, we can expect that these two sets coincide. The total complexity of the attack is $2^{512-s} + 2^{256+s}$ computations and 2^s memory.

Appendix A

The SHA-3 Competition

A.1 History of the SHA-3 Competition

The SHA-3 competition [105] has attracted a lot of attention in the cryptographic community. In 2006 NIST announced that it plans to replace the current hash function standard SHA-2 [136], with a more advanced function. The new standard was to be chosen through a public competition similar to the previous competition held for block ciphers [36], i.e. designers from all over the world can propose a new hash function, and after a thorough analysis one function will be chosen as a new standard. To efficiently sieve the weak candidates, NIST decided to have three rounds, each to last around a year, and in each round gradually the number of functions will be reduced by rejecting candidates with weaknesses.

A total of 64 teams submitted a candidate hash function to the NIST's open call in October 2008, and 51 function advanced to the first round (the other 13 proposals were refused due to incomplete or irregular submission packages). Most of the design teams came from academia, however, there were functions proposed by companies, as well as individuals. The proposed hash functions were based on various underlying transformations. A large group of functions appeared to be based on the block cipher AES – a primitive with a good security and efficiency[1]. Another large group of functions was based on simple software-efficient transforms – additions, rotations, XORS, and in some cases, shifts. The initial software comparison showed that practically these candidates are clear winners in speed, however, they were yet to withstand all possible attacks. The first round cryptanalysis led to a clear distinction between secure proposals and functions with some weaknesses. These weaknesses were ranging anywhere from hand-found collision or preimage attacks to highly theoretical attacks. Interestingly, the number of first-round secure proposals (i.e. the functions without weakness found) was approximately the same as the number of planned second round candidates – 14 hash functions advanced to this round in July 2009. The second round cryptanalysis saw the introduction of distinguishers for hash/compression functions. The candidates had stronger security level, hence the analysis had to evolve to attacks that show that (usually) the compression function can be distinguished from a random function. In December 2010, NIST has chosen 5 candidates for the last, third round. It is worth noticing that for each of the final round candidates, there exists an attack that shows some type of a

[1] It was already known that Intel plans to implement special instructions for the round transformation of AES in the upcoming processors, hence improving ten fold the efficiency of AES in software.

weakness on at least half rounds of the compression function. NIST plans to choose a winner in 2012.

A.2 The Hash Function LUX

The hash family LUX [110] was submitted to the SHA-3 competition. However, a weaknesses was found that led to collision, preimage attacks and slide distinguishers [142, 129, 116, 44]. The attacks were based on a weakness in the output phase of LUX, as well as the self similarity of the rounds. Further, we propose a simple tweak that resists all the previous attacks. First, we introduce a round counter (in a form of an XOR of a block index) thus making the rounds distinct, and discard the output phase (but increase the number of rounds in the blank-round phase).

A.2.1 Specifications

LUX is a byte oriented stream based hash function. It supports 224, 256, 384 and 512 bit digests and can hash a message of length up to 2^{64} bits. The only parameter for the different digests is m which represents the size in the matrices (the number of rows in the matrices) used for the internal representation of the state of the function. For LUX-224 and LUX-256 the value of m is 4, while for LUX-384 and LUX-512 it is 8.
LUX has an internal state S of $m \times 24$ bytes which can be divided into two parts:

- the buffer (B), which is a matrix of $m \times 16$ bytes, denoted $B_{i,j}$ where $0 \leq i \leq m-1$ and $0 \leq j \leq 15$
- the core (C), which is a matrix of $m \times 8$ bytes, denoted $C_{i,j}$ where $0 \leq i \leq m-1$ and $0 \leq j \leq 7$.

The internal state is manipulated by the state update function. The message is processed by small chunks of m bytes each.

Digest size	Message block (in bytes)	Core (in bytes)	Buffer (in bytes)	Internal state (in bytes)
224	4	4×8	4×16	96
256	4	4×8	4×16	96
384	8	8×8	8×16	192
512	8	8×8	8×16	192

Table A.1: Parameters for different digests.

A.2.1.1 State Update Function

Important building block of LUX is the state update function Φ. It takes the current state S (buffer+core) and a message block M_t of m bytes, and the block index t and produces a new state, i.e. :

$$S_{new} \leftarrow \Phi(S_{old}, M_t, t).$$

This process is defined as *one round*. LUX is little endian oriented. The input message is divided into chunks of m bytes. These chunks are treated as little endian words, i.e. least significant byte comes first. The state update function Φ itself can be decomposed into several consecutive steps. Let $B_{i,j} \in GF(2^8)$ be the elements of the buffer, and $C_{i,j} \in GF(2^8)$ be elements of the core. Let us denote by B_i the i-th column of the matrix B, i.e. $B_i = (B_{0,i}, B_{1,i}, \ldots, B_{m-1,i})^T$. Similarly, $C_i = (C_{0,i}, C_{1,i}, \ldots, C_{m-1,i})^T$. Let $M_t = (M_{0,t}, M_{1,t}, \ldots, M_{m-1,t})^T$ be a message block of m bytes. With "$\oplus$" we will denote vector XOR addition. Then the state update function, sequentially, does the following steps:

State update function Φ

Input $B = B_0||B_1||\ldots||B_{15} \quad C = C_0||C_1||\ldots||C_7$

1. **Add the message block to both the buffer and the core.**
   $B_0 \leftarrow B_0 \oplus M_t$
   $C_0 \leftarrow C_0 \oplus M_t$
2. **Update the buffer and the core separately.**
   $B \leftarrow F(B)$
   $C \leftarrow G(C)$
3. **Add the core to the buffer.**
   for $i = 0$ to 7 do
   $\quad B_{i+4} \leftarrow B_{i+4} \oplus C_i$
4. **Feedforward column of the buffer to the core**.
   $C_7 \leftarrow C_7 \oplus B_{15}$

Output B,C

In the step 2 of Φ we use functions that transform the buffer and the core. Let us define these remaining functions F and G.
The function F is used to manipulate only the buffer B. It is a simple cyclic rotation of the matrix by one column to the right. Therefore $F(B)$, in pseudo code, can be defined as:

Buffer function F

Input $B = B_0||B_1||\ldots||B_{15}$

for $i = 0$ to 15 do
$\quad \widetilde{B}_{(i+1)mod\ 16} \leftarrow B_i$

Output $\widetilde{B}$

The function G is one round of Rijndael where the core is seen as the state of Rijndael. This round transformation consists of SubBytes, ShiftRows, MixColumns, and AddConstant.

Core function G

Input C

$C \leftarrow \mathsf{SubBytes}(C)$
$C \leftarrow \mathsf{ShiftRows}(C)$

$C \leftarrow \mathsf{MixColumns}(C)$
$C \leftarrow \mathsf{AddConstant}(C)$

Output C

Although these four transformations are well known from Rijndael, below we will give a brief description of each of the transformations.

SubBytes. This non-linear byte-wise function is defined exactly as in the Rijndael specifications, i.e. the S-box used in SubBytes is the same as the S-box used in Rijndael.

$$S(X) = Y.$$

Let $X = X_1||X_2$, where X_1 are the first four bits of the byte X and X_2 the last four bits of X. Then the S-box used in LUX can be defined as:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

ShiftRows. This transformation cyclically rotates to the left, independently, each row of the core matrix. For LUX-224 and LUX-256, whose core matrix has four rows, the rotation constants are 0,1,3, and 4. Let us denote this vector by v, i.e. $v = (v_1, v_2, v_3, v_4) = (0, 1, 3, 4)$[2]. Then ShiftRows can be defined as:

$$C_{i,j} \leftarrow C_{i,(j+v_j) mod\ 8}$$

LUX-384 and LUX-512 have core matrix with eight rows, hence the rotation vector has eight coordinates: $v = (0, 1, 2, 3, 4, 5, 6, 7)$.

MixColumns. The MixColumns operation processes each column of the core matrix independently. A column is treated as an m-element vector in GF(2^8) and is multiplied by a matrix in GF(2^8). The multiplication is performed modulo the irreducible polynomial[3] $m(x) = x^8 + x^4 + x^3 + x + 1$. For LUX-224 and LUX-256 the matrix that defines this linear

[2] As in Rijndael-256.

[3] As in Rijndael.

transformation is the following[4]:

$$C_i^{\text{new}} \leftarrow A \cdot C_i^{\text{old}}; \quad A = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

For LUX-384 and LUX-512 the following matrix[5] is used:

$$A = \begin{pmatrix} 01 & 04 & 01 & 01 & 02 & 0c & 06 & 08 \\ 08 & 01 & 04 & 01 & 01 & 02 & 0c & 06 \\ 06 & 08 & 01 & 04 & 01 & 01 & 02 & 0c \\ 0c & 06 & 08 & 01 & 04 & 01 & 01 & 02 \\ 02 & 0c & 06 & 08 & 01 & 04 & 01 & 01 \\ 01 & 02 & 0c & 06 & 08 & 01 & 04 & 01 \\ 01 & 01 & 02 & 0c & 06 & 08 & 01 & 04 \\ 04 & 01 & 01 & 02 & 0c & 06 & 08 & 01 \end{pmatrix}$$

AddConstant. In hash functions usually constant addition is introduced in order to stop various slide attacks. Rijndael transformation is very sensitive to symmetric inputs. If all the elements of the core matrix C are the same, then the previous three transformations will produce a new state with, again, all elements equal to each other. To destroy this unwanted property we use the fourth transformation AddConstant. It is a simple XOR of $0x2ad01c64 \oplus BI$ to the core column C_1. This constant corresponds to the first 8 hexadecimal digits of e. The value BI denotes the block index (starting from 1 for the first message block). In the blank round phase, the block index takes the value of 0.

AddConstant
Input C
$C_1 \leftarrow C_1 \oplus 0x2ad01c64 \oplus BI$
Output C

A.2.1.2 Initializations and padding

The buffer and the core are initialized by setting all the elements equal to zero. The standard padding procedure is applied to the message,. First, the message is divided into message blocks of m bytes ($8m$ bits). If the length of the last message block is less than $8m$ bits then this block is padded with one, followed by number of zeros such that the block length becomes equal to $8m$. If the last block has exactly $8m$ bits, than additional block is introduced, with one followed by $8m-1$ zeros. For LUX-224 and LUX-256, at the end two additional 4-byte blocks, containing the binary expression of the length of the non-padded message, are created. For LUX-384 and LUX-512 only one 8-byte block with the message length is created. This padded message becomes the input message for the hash function.

[4] As in Rijndael.
[5] As in Grindahl-512.

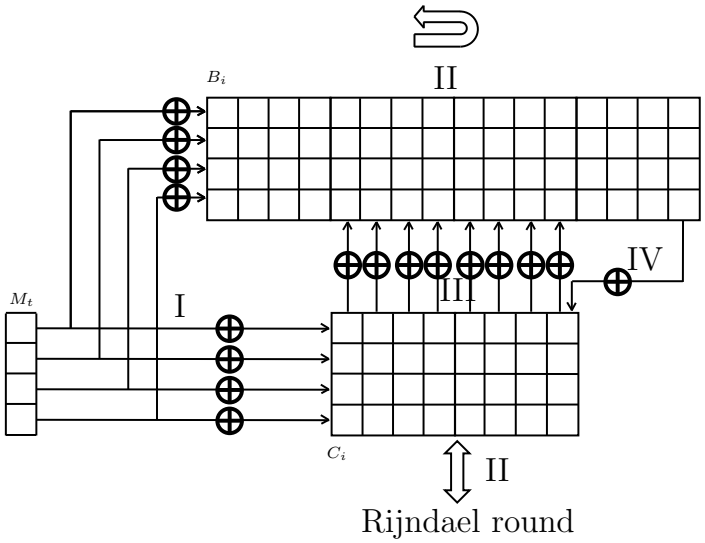

Figure 1: The round function of LUX. I - message addition to the buffer and the core. II - update of the buffer and the core. III - Addition of the core to the buffer. IV - addition of one column from the buffer to the core.

A.2.1.3 Hashing and output

The hash function LUX, similarly to the other stream based functions, uses the principle input message / blank rounds and output hash. These two phases are executed one after another. Depending on the message length and the size of the digest, the first phase has different number of rounds.

Input phase After the state is being initialized and the message is being padded the input message phase starts. In this phase the whole message, block by block, is "absorbed", without outputting anything. Each message block, starting from the first one, is passed as an argument to the state update function Φ which transforms the internal state, in one round, to a new state. The input phase ends when, sequentially, all message blocks are processed. Obviously the number of rounds of this phase depends only on the size of the message, i.e. the number of message blocks in the padded message.

Blank rounds phase and producing outputs Blank rounds phase is typically used to increase the diffusion of the last message blocks. For LUX, this phase consists of 32 rounds. In each of these rounds the input message block as well as the block index are considered to be zeros.

After the blank round phase is finished, the words of the core are taken as output. For LUX-256 and LUX-512 all columns of the core ($8 \cdot 32 = 256, 8 \cdot 64 = 512$), for LUX-224 the 7 left-most columns ($7 \cdot 32 = 224$), while for LUX-384 the 6 left-most ($6 \cdot 64 = 384$) columns of the core are the hash value of the message.

From the algorithmic point of view, these two phases do not differ because they are

	Core	Buffer
LUX-256	4×8	4×16
LUX_{v1}-256	4×8	4×12
LUX_{v2}-256	4×4	4×16
LUX_{v3}-256	4×4	4×12

Table A.2: Reduced variants of LUX-256.

divided into rounds and in each round the state update function gets the three necessary parameters: the message block, the block index and the internal state. Therefore the same function Φ can be used in both of the phases. Let the padded message consists of k blocks. Then the hash function LUX can be defined as:

Hash function LUX-256

Input $M = M_1||M_1||\ldots||M_k$

for i=1 to k
    $S \leftarrow \Phi(S, M_i, i)$
for i=1 to 32
    $S \leftarrow \Phi(S, 0, 0)$
output $C_1, \ldots, C_8$

A.2.1.4 Reduced Variants

The block based hash function usually takes a high number of rounds to process one message block. By decreasing this number of rounds, reduced variants of the function are produced. In the analysis of the hash functions it is common to analyze these variants if the original function seems to be resistant to attacks. Since LUX uses different design principle (stream based), obviously, such variants are not possible. Yet, we can propose an alternative with decreased size of the internal state. Since the internal state of LUX consists of a core and a buffer, we propose a variants whose core and/or buffer have less number of columns. Although we claim it is critical to have a buffer that has 4 columns more than the core, in the front and at the end (in the passive buffer), yet it will be interesting to analyze how the security margin behaves for the variants with less number of columns in the buffer and the core. Reduced internal state variants of LUX-256 are proposed in Table A.2.

Different variants can be made by reducing the number of blank rounds, from 16 rounds to 12 or 8 blank rounds. In the output hash phase in each round one column of output is produced. A variant, where in each round 2 or 4 columns of output are produced, can also be analyzed.

We recommend these variants for the research on security margins of LUX but not for actual use.

Table A.3: Security levels for LUX.

	Attack Complexity			
Digest	224	256	384	512
Collision attack	2^{112}	2^{128}	2^{192}	2^{256}
Preimage attack	2^{224}	2^{256}	2^{384}	2^{512}
Second preimage attack	2^{224}	2^{256}	2^{384}	2^{512}
HMAC-PRF distinguisher	2^{112}	2^{128}	2^{192}	2^{256}

A.2.1.5 HMAC and PRF

The LUX hash functions are suitable for a standard construction of HMAC:

$$\mathrm{HMAC}_K(m) = h\big((K + c_1)||h((K + c_2)||m)\big).$$

The secret key K is padded with extra zeros to have a minimal length divisible by $4m$ ($8m$ if LUX-384 or LUX-512 is used). The constants have the following values: $c_1 = 0\text{x}5c5c\ldots5c$, $c_2 = 0\text{x}3636\ldots36$. The inputs $K + c_1, K + c_2$ are divided into message blocks of m bytes and then processed by the hash function.

LUX can be used as a base of pseudo-random function (PRF) under different constructions, e.g., as HMAC-PRF.

A.2.1.6 Security levels

The security levels, claimed for the hash functions of the family LUX, are presented in Table A.3. Also, any m-bit subset of the output bits, meets the required levels for collision resistance and preimage resistance of $2^{m/2}$ and 2^m respectively.

A.3 The SHA-3 Competition: Lessons Learned (or How to Design a Secure Hash Function)

The crypto group at the University of Luxembourg has taken an active part in the analysis of a dozen SHA-3 candidates: Abacus [111], Blue Midnight Wish [59], Boole [80], DCH [75], Edon-R[79, 80], EnRUPT[80], Hamsi [107], NaSHA [112], Sarmal [80], Skein [77, 78], StreamHash [76] and others. We would like to share the experience we gathered through these attacks, as well as the analysis provided by other researchers we would like to raise the following challenge: how to build a secure hash function? To answer this question, one has to start from the definition of a secure hash. Indeed, the SHA-3 analysis has taught us that we should first focus on the security of the underlying compression function[6]. To be secure this function has to withstand all possible collision and (second) preimage attacks as well as all known (and possibly yet to be invented) distinguishers. For none of these attacks there exist a formal design method that stops the attacks. Hence, to answer the above question – it is not possible to build a secure function. However, we can try to build the next best thing – a function arguably resistant to all known attacks. In other words, under certain reasonable

[6]Or in the case of the sponge construction–the underlying permutation.

assumptions, we can *informally* prove that our function does not experience any known weaknesses. To do so, we have to take into account one-by-one each attack technique, find the weakness property used by the technique, gather all possible theory on how to stop the attack, and implement it in the function. The further recommendations can be summarized as follows:

1. Choose a hash function mode that has provably secure properties
2. To stop various generic attacks, such as the herding attack and the long second-preimage attack, construct a compression function that is either a wide-pipe or a single pipe, but with a salt and a block index. If the latter variant is used, then check the security of the function under the assumption that the attacker fully controls the values of the block index and the salt
3. The compression function has to have a high security margin against differential attacks, i.e. prove that the best differential trail covers only a fraction of the total number of rounds
4. If the compression function is invertible, than make sure that the state is larger than $2n + m$, where m is the size of the input message block
5. Use constants to stop slide and rotational attacks
6. The algebraic degree of the function has to be sufficiently high. Otherwise various algebraic attacks, e.g. cube attacks, can be launched

A.3.1 A Secure Hash Function Mode

The mode of a hash function provides a method of turning the fixed input length compression function into an arbitrary input length hash function. If the mode used in the hash has severe weaknesses, then regardless of the compression function, the hash function will be insecure. Therefore, the first and foremost thing is choosing a proper hash mode. There are several modes that were proven the be secure when the underlying compression function is secure. Merkle-Damgård construction [100, 45] used in all of the hash functions from MD4 to SHA-2, the sponge [15] used when the compression function is wide-pipe permutation, and the tree-based mode [50] which provides the opportunity of computing in parallel the compression functions, were all proven to be secure modes. There exist some other ad-hoc modes, however, using unproven modes is not recommended. Therefore, the designer is advised to use one of these modes. In case a new mode is proposed it is crucial to prove the security of the mode.

Further, we assume that the chosen hash mode is secure. Hence, we can proceed with the design (and analysis) of the compression function only.

A.3.2 The Choice of Compression Function

Most of the compression functions, with a few exceptions, are based on a permutation. To make the compression function non-invertible, after the last round of the permutation there is some type of a feedforward, and the previous chaining value or/and the message is added to the state (thus making the permutation a function). A list of secure constructions applying the idea of feedforward is given in [118]. The compression function does not necessarily have to be a function, i.e. it can stay a permutation, however then the internal state of the permutation has to be at least $2n$ bits in order to stop possible preimage attacks. The double-pipe (or wider) functions in general are more resistant to generic attacks (due to the high complexity of finding internal collisions), however, we would not like to give any preference

– we assume that single-pipe constructions (with a careful design strategy) can be as secure as the double-pipe[7].

Standard inputs to the compression function are the input chaining value and the message block. Several SHA-3 candidates followed the HAIFA design strategy [19], and took as additional inputs a salt and a block counter thus preventing several generic attacks on the Merkle-Damgård [73, 47, 71]. Using the salt and the block index in the compression function is advantageous and desirable. In analysis, the salt and the block index are assumed to be regular input variables (just as the message and the chaining value), the attacker can manipulate their values, hence a special attention has to be taken to assure that they do not introduce any weakness in the function. For example, XORing the salt (or the block index) to the input message block or the chaining value, immediately leads to a trivial attack: given the same difference in the salt and the message, the XOR will produce zero difference, hence a pseudo-collision attack can be launched.

A.3.3 Collision Resistance

The collision attacks on modern compression functions have been focused on finding a differential trail in the function that ends with a zero difference (or with a small Hamming weight difference, leading to a near collision). This does not have to be the case for all compression functions. Examples that contradict this assumption are the functions based on some hard mathematical problems. Nevertheless, almost all compression functions are composed of rounds of simple transforms (rather than structural transforms that mimic some mathematical function), and therefore the search of differential trails for collisions stays the main challenge. The trails are important as well for finding differential distinguishers, hence their probability can be as low as $2^{-(n-1)}$. So far, these trails were found exclusively by hand since the big state of the compression function prohibits feasible automatic search.

Only a few SHA-3 candidates had a clear analysis and estimated the probability of the best differential trails. Interestingly, they were based on unkeyed permutations, i.e. the message is injected in the state at the beginning of the compression function followed by a number of similar round transforms (without any input from the message). This significantly reduces the search space – it becomes sufficient to find the probability of the best trail only for the unkeyed round transforms. This is done by proving the upper bound on the probability of the best round-reduced trail, i.e. if the best trail on t rounds has a probability at most p, then the trail on $t \cdot k$ rounds has probability at most p^k. When, p is sufficiently low, then $p^k \ll 2^{-n}$, giving a high security margin. If one tries to apply the same approach to keyed permutations, i.e. permutations with message injections in the rounds, he will usually fail since the probability p of the round-reduced trails is often very high (and therefore $p^k > 2^{-n}$). One can try to take more rounds in the round-reduced trail, i.e. to increase t, which will result in lower probability p. However, finding the upper bound p on the probability of the best trail on a large number of rounds t is often impossible.

The problem of finding the upper bounds on the probabilities usually can be solved for byte-oriented substitution-permutation (SP) networks. A technique for building SP primitives resistant to differential (and linear) attacks is given in [42]. For byte-oriented compression function, in general, the search for the best (round-reduced) trails is easier (and usually feasible), particulary for unkeyed permutations. With a brute force (or by taking into account the properties of the linear layer, such as the branch number), it is possible to find the minimal number of active S-boxes in a round-reduced trail and from there to compute the minimal number of the active S-boxes for the full-round function. However, what most

[7] Each of the five NIST third round candidates is a wide-pipe construction.

of the SHA-3 designers seem to underestimate is the power of the message modification technique. They found the upper bound on the probability of the best trail, but did not take into account the possibility that some rounds can be passed with probability 1 (instead of the probability defined by the trail), since the attacker has the full freedom to choose all inputs and to fix some values of the internal state. For many of the SHA-3 function the rebound attack, exploiting the idea of message modification, was fatal. Therefore, when proving the resistance of a function against differential attacks, the designer has to keep in mind that some number of rounds can be passed for free, and hence he should provide much higher security margin than the total number of rounds. Unfortunately, it is still not clear how high the margin has to be (even for the above functions) as it is hard to estimate the number of "free" rounds. If t rounds are required for a complete difusion in the state, then a recommendation would be to have at least $3t$ rounds margin, i.e. if the function has r rounds, then the designer has to prove that a function with $r + 3t$ rounds is resistant against differential attacks.

A.3.4 Preimage Resistance

Preimage and second-preimage attacks are ways to exploit the possibility of inverting the compression function and launching a meet-in-the-middle attack. Based on this property we can divide the functions into two categories: invertible and non-invertible.

The invertible compression functions are basically a permutation: keyed or unkeyed. Invertible functions based on a keyed permutation are rare (for their preimage resistance refer to the following paragraph on non-invertible compression functions), further we analyze only unkeyed permutations. As mentioned above, in this type of function, at the beginning of the permutation, the message block is XORed to the state, followed by rounds of transformations. Since the function is invertible, a necessary requirement for the permutation is to have a state of at least $2n$ bits. Otherwise, a simple meet-in-the-middle attacks would make the preimage search trivial. However, a more detailed analysis shows that the size of the state actually should depend on the size of the injected message block – if the message block has m bits, then the size of the state has to be at least $2n + m$ bits (for n-bit preimage security). If this is not the case, then again it is possible to launch a MITM attack on two consecutive compression functions. The $2n + m$-bit state is sufficient when the underlying permutation is ideal, i.e. it has no weaknesses on its own. Therefore, it is desirable to have some type of security margin obtained by taking a state with a size greater than $2n + m$ bits.

The non-invertible compression functions are usually based on keyed permutation with the additional feedforward after the last round. For this type of constructions the designer has to take into account the splice-and-cut technique for producing preimages. Therefore, the message schedule has to be chosen carefully. It should not be possible to split the set of expanded message words into two sets with some independent words. In general, when the state is double-pipe, the splice-and-cut technique should be inapplicable (since the MITM space becomes $2n$-bit wide, and therefore the MITM attack requires 2^n), but this is true in the cases when the function has no known weaknesses.

A.3.5 Resistance against Various Distinguishers

Now let us focus on possible distinguishers for the compression function. Note that the case of differential distinguishers has been covered in the section on collision resistance.

Slide attacks and rotational cryptanalysis are basically applicable to primitives that do not use constants (or use some very specific constants). To resist these attacks, the designer should preferably introduce constants in each round of the function. For each round, the

constants should be different and non-rotational – the best choice is to use some randomly generated values. The same constant should not be added (or XORed) to the same message word, otherwise, the effect of the constant can be canceled. In particular, in unkeyed permutations, the words of the state where the constant is added, should not be the same as the words where the message block is added.

The designs with a low algebraic degree are susceptible to various algebraic attacks. Aside from the ARX designs, where the high degree is ensured by the additions, the function in which the only non-linear transforms are in the form of small S-boxes (usually 4x4), or low-degree transformations (such as AND and OR), should be carefully examined to ensure that the degree is the highest possible. For example, when in n-bit permutation one can find some set of $m < n$ variables in some intermediate state such that in both directions all bits of the output are functions of degree strictly less than n in the variables chosen, then it is possible to launch a zero-sum distinguisher (see the work of Aumasson-Meyer [9]). Therefore, the designs with low-degree rounds, should have much higher number of rounds to resist the above attacks.

List of Publications in LNCS

1. Alex Biryukov, Ivica Nikolić, Search for Related-key Differential Characteristics in DES-like ciphers , FSE 2011.
2. Alex Biryukov, Ivica Nikolić, Arnab Roy, Boomerang Attacks on BLAKE-32, FSE 2011.
3. Dmitry Khovratovich, Ivica Nikolić, Christian Rechberger, Rotational Rebound Attacks on Reduced Skein, ASIACRYPT 2010. *(Best Paper Award)*
4. Ivica Nikolić, Josef Pieprzyk, Przemysław Sokołowski, Ron Steinfeld, Known and Chosen Key Differential Distinguishers for Block Cipher, ICISC 2010.
5. Ivica Nikolić, Tweaking AES, SAC 2010.
6. Alex Biryukov, Ivica Nikolić, Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others, EUROCRYPT 2010.
7. Dmitry Khovratovich, Ivica Nikolić, Rotational Cryptanalysis of ARX, FSE 2010.
8. Krystian Matusiewicz, María Naya-Plasencia, Ivica Nikolić, Yu Sasaki, Martin Schläffer, Rebound Attack on the Full Lane Compression Function, ASIACRYPT 2009.
9. Alex Biryukov, Dmitry Khovratovich, Ivica Nikolić, Distinguisher and Related-Key Attack on the Full AES-256, CRYPTO 2009.
10. Dmitry Khovratovich, Alex Biryukov, Ivica Nikolić, Speeding up Collision Search for Byte-Oriented Hash Functions, CT-RSA 2009.
11. Dmitry Khovratovich, Ivica Nikolić, Ralf-Philipp Weinmann, Meet-in-the-Middle Attacks on SHA-3 Candidates, FSE 2009.
12. Alex Biryukov, Praveen Gauravaram, Jian Guo, Dmitry Khovratovich, San Ling, Krystian Matusiewicz, Ivica Nikolić, Josef Pieprzyk, Huaxiong Wang, Cryptanalysis of the LAKE Hash Family, FSE 2009.
13. Ivica Nikolić and Alex Biryukov, Collisions for step-reduced SHA-256, FSE 2008.

List of Results

Table A.4: Summary of the results presented in the thesis

Name	Type	Summary	Section
AES-128	cipher	Found the best RK characteristic	9.2.1
		RK boomerang attack on 7 rounds	9.2.2
AES-192	cipher	Found the best RK characteristic	9.2.1
		Improved the RK boomerang attack on 12 rounds	9.2.3
AES-256	cipher	Found the best RK characteristic	9.2.1
Boole	hash	Found preimage attacks	12.2
BLAKE32	hash	Found boomerang attacks on up to 7 rounds	8.3
	cipher	Found RK boomerang attacks on up to 8 rounds	8.4
byte-Camellia	cipher	Found the best RK characteristic	9.3.1
		Launched chosen-key RK attack on full-round	9.3.2
DES	cipher	Found the best RK characteristic for some rounds	10.3
DESL	cipher	Found the best RK characteristic for some rounds	10.4
BMW	hash	Rotational distinguishers on (modified) CF	11.2
Edon-R	hash	Found preimage attacks	12.3
Khazad	cipher	Found the best RK characteristic	9.4.1
		RK boomerang attack on 7 rounds	9.4.2
		Launched chosen-key RK attack on full-round	9.4.3
LAKE	hash	Found (practical) pseudo collision attacks	7.2
s^2DES	cipher	Found the best RK characteristic	10.5
Sarmal	hash	Found preimage attacks	12.4
Skein-256	hash	Rotational distinguishers on 40 rounds for CF	11.1
Skein-512	hash	Rotational distinguishers on 44 rounds for CF	11.1
SHA-256	hash	Found practical collisions attacks on up to 25 rounds	6.3
Threefish	cipher	Rotational distinguishers for up to 42 rounds	11.1
xAES	cipher	Tweaked the key schedule of AES	
		and proved the resistance against RK attacks	9.5

Bibliography

[1] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang. Preimages for step-reduced SHA-2. In Matsui [94], pages 578–597.

[2] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. Camellia: A 128-bit block cipher suitable for multiple platforms - design and analysis. In D. R. Stinson and S. E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2000.

[3] K. Aoki, K. Kobayashi, and S. Moriai. Best differential characteristic search of FEAL. In Biham [17], pages 41–53.

[4] K. Aoki and Y. Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In Avanzi et al. [11], pages 103–119.

[5] J.-P. Aumasson. Practical distinguisher for the compression function of Blue Midnight Wish, 2009. Available at `http://131002.net/data/papers/Aum10.pdf`.

[6] J.-P. Aumasson, I. Dinur, W. Meier, and A. Shamir. Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In Dunkelman [51], pages 1–22.

[7] J.-P. Aumasson, J. Guo, S. Knellwolf, K. Matusiewicz, and W. Meier. Differential and invertibility properties of BLAKE. In Hong and Iwata [65], pages 318–332.

[8] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. SHA-3 proposal BLAKE. Submission to NIST, 2008.

[9] J.-P. Aumasson and W. Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. NIST mailing list.

[10] J.-P. Aumasson, W. Meier, and R. C.-W. Phan. The hash function family LAKE. In Nyberg [114], pages 36–53.

[11] R. M. Avanzi, L. Keliher, and F. Sica, editors. *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*. Springer, 2009.

[12] P. Barreto and V. Rijmen. The Anubis Block Cipher. In *Submission to the NESSIE Project*, 2000.

[13] P. Barreto and V. Rijmen. The Khazad Legacy-Level Block Cipher. In *Submission to the NESSIE Project*, 2000.

[14] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[15] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. On the indifferentiability of the Sponge construction. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

[16] E. Biham. New types of cryptanalytic attacks using related keys. *J. Cryptology*, 7(4):229–246, 1994.

[17] E. Biham, editor. *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*. Springer, 1997.

[18] E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In J. Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

[19] E. Biham and O. Dunkelman. A framework for iterative hash functions HAIFA. NIST Second Cryptographic Hash Workshop, Santa Barbara, August 2006.

[20] E. Biham, O. Dunkelman, and N. Keller. Related-key boomerang and rectangle attacks. In Cramer [40], pages 507–525.

[21] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. In A. Menezes and S. A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[22] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.

[23] E. Biham and A. Shamir. Differential cryptanalysis of the full 16-round DES. In E. F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992.

[24] A. Biryukov. Analysis of involutional ciphers: Khazad and Anubis. In T. Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 45–53. Springer, 2003.

[25] A. Biryukov, editor. *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007.

[26] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir. Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds. In Gilbert [55], pages 299–319.

[27] A. Biryukov, P. Gauravaram, J. Guo, D. Khovratovich, S. Ling, K. Matusiewicz, I. Nikolić, J. Pieprzyk, and H. Wang. Cryptanalysis of the LAKE hash family. In Dunkelman [51], pages 156–179.

[28] A. Biryukov and D. Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In Matsui [94], pages 1–18.

[29] A. Biryukov, D. Khovratovich, and I. Nikolić. Distinguisher and related-key attack on the full AES-256. In S. Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 231–249. Springer, 2009.

[30] A. Biryukov and I. Nikolić. Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In Gilbert [55], pages 322–344.

[31] A. Biryukov and I. Nikolić. Search for related-key differential characteristics in DES-like ciphers. In *FSE*, 2011.

[32] A. Biryukov, I. Nikolić, and A. Roy. Boomerang attacks on BLAKE-32. In *FSE*, 2011.

[33] A. Biryukov and A. Shamir. Structural cryptanalysis of SASAS. In B. Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2001.

[34] A. Biryukov and D. Wagner. Slide attacks. In Knudsen [84], pages 245–259.

[35] G. Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[36] W. E. Burr. Selecting the Advanced Encryption Standard. *IEEE Security & Privacy*, 1(2):43–52, 2003.

[37] C. D. Cannière and C. Rechberger. Finding SHA-1 characteristics: General results and applications. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[38] F. Chabaud and A. Joux. Differential collisions in SHA-0. In H. Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.

[39] D. R. Chowdhury, V. Rijmen, and A. Das, editors. *Progress in Cryptology - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings*, volume 5365 of *Lecture Notes in Computer Science*. Springer, 2008.

[40] R. Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

[41] J. Daemen, L. R. Knudsen, and V. Rijmen. The block cipher Square. In Biham [17], pages 149–165.

[42] J. Daemen and V. Rijmen. The wide trail design strategy. In B. Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001.

[43] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[44] W. Dai. Official comment: LUX. NIST mailing list (local link), 2009.

[45] I. Damgård. A design principle for hash functions. In Brassard [35], pages 416–427.

[46] M. Daum. *Cryptanalysis of Hash Functions of the MD4-Family*. PhD thesis, Ruhr-Universität Bochum, May 2005.

[47] R. D. Dean. Formal aspects of mobile code security. PhD thesis, Princeton University, January 1999.

[48] W. Diffie and M. E. Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10:74–84, June 1977.

[49] I. Dinur and A. Shamir. Cube attacks on tweakable black box polynomials. In Joux [68], pages 278–299.

[50] Y. Dodis, L. Reyzin, R. L. Rivest, and E. Shen. Indifferentiability of permutation-based compression functions and tree-based modes of operation, with applications to MD6. In Dunkelman [51], pages 104–121.

[51] O. Dunkelman, editor. *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*. Springer, 2009.

[52] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting. Improved cryptanalysis of Rijndael. In Schneier [130], pages 213–230.

[53] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein hash function family. In *Submitted to SHA-3 Competition*, 2008.

[54] P.-A. Fouque, G. Leurent, and P. Nguyen. Automatic search of differential path in MD4. *Cryptology ePrint Archive, Report 2007/206*, 2007. Available at `http://eprint.iacr.org/2007/206.pdf`.

[55] H. Gilbert, editor. *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*. Springer, 2010.

[56] H. Gilbert and H. Handschuh. Security analysis of SHA-256 and sisters. In M. Matsui and R. J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 175–193. Springer, 2003.

[57] H. Gilbert and M. Minier. A collision attack on 7 rounds of Rijndael. In *AES Candidate Conference*, pages 230–241, 2000.

[58] H. Gilbert and T. Peyrin. Super-Sbox cryptanalysis: Improved attacks for AES-like permutations. In Hong and Iwata [65], pages 365–383.

[59] D. Gligoroski, V. Klima, S. J. Knapskog, M. El-Hadedy, J. Amundsen, and S. F. Mjølsnes. Cryptographic Hash Function BLUE MIDNIGHT WISH. *Submission to NIST (Round 1)*, 2008. Available at `http://people.item.ntnu.no/~danilog/Hash/BMW/Supporting_Documentation/BlueMidnightWishDocumentation.pdf`.

[60] D. Gligoroski, V. Klima, S. J. Knapskog, M. El-Hadedy, J. Amundsen, and S. F. Mjølsnes. Cryptographic Hash Function BLUE MIDNIGHT WISH. *Submission to NIST (Round 2)*, 2009. Available at `http://people.item.ntnu.no/~danilog/Hash/BMW-SecondRound/Supporting_Documentation/BlueMidnightWishDocumentation.pdf`.

[61] D. Gligoroski, R. S. Ødegård, M. Mihova, S. J. Knapskog, L. Kocarev, and A. Drápal. Cryptographic hash function Edon-R, `http://people.item.ntnu.no/~danilog/Hash/Edon-R/Supporting\_Documentation/EdonRDocumentation.pdf`. Submission to NIST, 2008.

[62] M. Gorski and S. Lucks. New related-key boomerang attacks on AES. In Chowdhury et al. [39], pages 266–278.

[63] J. Guo and S. S. Thomsen. Distinguishers for the compression function of Blue Midnight Wish with probability 1. In *Selected Areas of Cryptography*, 2010.

[64] P. Hawkes, M. Paddon, and G. G. Rose. On corrective patterns for the SHA-2 family. Cryptology eprint Archive, August 2004.

[65] S. Hong and T. Iwata, editors. *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *Lecture Notes in Computer Science*. Springer, 2010.

[66] S. Indesteege, F. Mendel, B. Preneel, and C. Rechberger. Collisions and other non-random properties for step-reduced SHA-256. In Avanzi et al. [11], pages 276–293.

[67] L. Ji and X. Liangyu. Attacks on round-reduced BLAKE. Cryptology ePrint Archive, Report 2009/238, 2009. Avalaible at `http://eprint.iacr.org/2009/238.pdf`.

[68] A. Joux, editor. *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*. Springer, 2009.

[69] A. Joux and T. Peyrin. Hash functions and the (amplified) boomerang attack. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2007.

[70] P. Junod and S. Vaudenay. FOX : A new family of block ciphers. In H. Handschuh and M. A. Hasan, editors, *Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2004.

[71] J. Kelsey and T. Kohno. Herding hash functions and the Nostradamus attack. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.

[72] J. Kelsey, T. Kohno, and B. Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In Schneier [130], pages 75–93.

[73] J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In Cramer [40], pages 474–490.

[74] D. Khovratovich, A. Biryukov, and I. Nikolić. Speeding up collision search for byte-oriented hash functions. In M. Fischlin, editor, *CT-RSA*, volume 5473 of *Lecture Notes in Computer Science*, pages 164–181. Springer, 2009.

[75] D. Khovratovich and I. Nikolić. Cryptanalysis of DCH-n, 2008. Available at `https://cryptolux.org/mediawiki/uploads/1/17/DCH--analysis.pdf`.

[76] D. Khovratovich and I. Nikolić. Cryptanalysis of StreamHash, 2009. Available at `https://cryptolux.org/mediawiki/uploads/e/e9/StreamHash-analysis.pdf`.

[77] D. Khovratovich and I. Nikolić. Rotational cryptanalysis of ARX. In Hong and Iwata [65], pages 333–346.

[78] D. Khovratovich, I. Nikolić, and C. Rechberger. Rotational rebound attacks on reduced Skein. In M. Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2010.

[79] D. Khovratovich, I. Nikolić, and R.-P. Weinmann. Cryptanalysis of Edon-R, 2008. Available at `https://cryptolux.org/mediawiki/uploads/2/2c/Edon-analysis.pdf`.

[80] D. Khovratovich, I. Nikolić, and R.-P. Weinmann. Meet-in-the-middle attacks on SHA-3 candidates. In Dunkelman [51], pages 228–245.

[81] J. Kim, S. Hong, and B. Preneel. Related-key rectangle attacks on reduced AES-192 and AES-256. In Biryukov [25], pages 225–241.

[82] K. Kim. Construction of DES-like S-boxes based on boolean functions satisfyieng the SAC. In H. Imai, R. L. Rivest, and T. Matsumoto, editors, *ASIACRYPT*, volume 739 of *Lecture Notes in Computer Science*, pages 59–72. Springer, 1991.

[83] L. R. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

[84] L. R. Knudsen, editor. *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*. Springer, 1999.

[85] L. R. Knudsen and V. Rijmen. Known-key distinguishers for some block ciphers. In K. Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 315–324. Springer, 2007.

[86] X. Lai and J. L. Massey. Hash function based on block ciphers. In R. A. Rueppel, editor, *EUROCRYPT*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 1992.

[87] M. Lamberger, F. Mendel, C. Rechberger, V. Rijmen, and M. Schläffer. Rebound distinguishers: Results on the full Whirlpool compression function. In Matsui [94], pages 126–143.

[88] G. Leander, C. Paar, A. Poschmann, and K. Schramm. New lightweight DES variants. In Biryukov [25], pages 196–210.

[89] S. Lee, S. Hong, S. Lee, J. Lim, and S. Yoon. Truncated differential cryptanalysis of Camellia. In K. Kim, editor, *ICISC*, volume 2288 of *Lecture Notes in Computer Science*, pages 32–38. Springer, 2001.

[90] J. Lu, O. Dunkelman, N. Keller, and J. Kim. New impossible differential attacks on AES. In Chowdhury et al. [39], pages 279–293.

[91] J. Lu, J. Kim, N. Keller, and O. Dunkelman. Improving the efficiency of impossible differential cryptanalysis of reduced Camellia and MISTY1. In T. Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 370–386. Springer, 2008.

[92] M. Matsui. Linear cryptoanalysis method for DES cipher. In T. Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[93] M. Matsui. On correlation between the order of S-boxes and the strength of DES. In A. D. Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 1994.

[94] M. Matsui, editor. *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*. Springer, 2009.

[95] F. Mendel, T. Peyrin, C. Rechberger, and M. Schläffer. Improved cryptanalysis of the reduced Grøstl compression function, ECHO permutation and AES block cipher. In M. J. J. Jr., V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 16–35. Springer, 2009.

[96] F. Mendel, N. Pramstaller, C. Rechberger, M. Kontak, and J. Szmidt. Cryptanalysis of the GOST hash function. In D. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 162–178. Springer, 2008.

[97] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. Analysis of step-reduced SHA-256. In Robshaw [121], pages 126–143.

[98] F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. The Rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In Dunkelman [51], pages 260–276.

[99] F. Mendel and M. Schläffer. Collisions for round-reduced LAKE. In Mu et al. [102], pages 267–281.

[100] R. C. Merkle. One way hash functions and DES. In Brassard [35], pages 428–446.

[101] H. Morita, K. Ohta, and S. Miyaguchi. A switching closure test to analyze cryptosystems. In *CRYPTO 1991*, volume 576 of *LNCS*, pages 183–193. Springer, 1992.

[102] Y. Mu, W. Susilo, and J. Seberry, editors. *Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia, July 7-9, 2008, Proceedings*, volume 5107 of *Lecture Notes in Computer Science*. Springer, 2008.

[103] F. Muller. A new attack against Khazad. In C.-S. Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 347–358. Springer, 2003.

[104] National Bureau of Standards. Data Encryption Standard. U.S. Department of Commerce, FIPS pub. 46, January 1977.

[105] National Institute of Standards and Technology. Cryptographic hash algorithm competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[106] National Institute of Standards and Technology. Advanced encryption standard (AES). FIPS 197, November 2001.

[107] I. Nikolić. Near collisions for the compression function of Hamsi-256. In *CRYPTO rump session*, 2009. Available at https://cryptolux.org/mediawiki/uploads/9/95/Hamsi.pdf.

[108] I. Nikolić. Tweaking AES. In *Selected Areas of Cryptography*, 2010.

[109] I. Nikolić and A. Biryukov. Collisions for step-reduced SHA-256. In Nyberg [114], pages 1–15.

[110] I. Nikolić, A. Biryukov, and D. Khovratovich. Hash family LUX - algorithm specifications and supporting documentation. Submission to NIST, 2008.

[111] I. Nikolić and D. Khovratovich. Second preimage attack on Abacus, 2008. Available at https://cryptolux.org/mediawiki/uploads/f/fa/Abacus-analysis.pdf.

[112] I. Nikolić and D. Khovratovich. Free-start attacks on NaSHA, 2009. Available at https://cryptolux.org/mediawiki/uploads/a/a2/NaSHA-analysis.pdf.

[113] I. Nikolić, J. Pieprzyk, P. Sokołowski, and R. Steinfeld. Rotational cryptanalysis of (modified) versions of BMW and SIMD. Available at https://cryptolux.org/mediawiki/uploads/0/07/Rotational_distinguishers_%28Nikolic%2C_Pieprzyk%2C_Sokolowski%2C_Steinfeld%29.pdf, 2010.

[114] K. Nyberg, editor. *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*. Springer, 2008.

[115] S. Paul and B. Preneel. Solving systems of differential equations of addition. In C. Boyd and J. M. G. Nieto, editors, *ACISP*, volume 3574 of *Lecture Notes in Computer Science*, pages 75–88. Springer, 2005.

[116] T. Peyrin. Slide attacks on LUX. NIST mailing list (local link), 2008.

[117] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, Jan. 1993.

[118] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In D. R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 1993.

[119] J.-J. Quisquater and J.-P. Delescaille. How easy is collision search? Application to DES (extended summary). In *EUROCRYPT 1989*, volume 434 of *LNCS*, pages 429–434, 1989.

[120] J.-J. Quisquater and J.-P. Delescaille. How easy is collision search. New results and applications to DES. In Brassard [35], pages 408–413.

[121] M. J. B. Robshaw, editor. *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*. Springer, 2006.

[122] G. G. Rose. Design and primitive specification for Boole. `http://seer-grog.net/BoolePaper.pdf`.

[123] Y. Sasaki and K. Aoki. A preimage attack for 52-step HAS-160. In P. J. Lee and J. H. Cheon, editors, *ICISC*, volume 5461 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2008.

[124] Y. Sasaki and K. Aoki. Preimage attacks on 3, 4, and 5-pass HAVAL. In J. Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271. Springer, 2008.

[125] Y. Sasaki and K. Aoki. Preimage attacks on step-reduced MD5. In Mu et al. [102], pages 282–296.

[126] Y. Sasaki and K. Aoki. Finding preimages in full MD5 faster than exhaustive search. In Joux [68], pages 134–152.

[127] Y. Sasaki and K. Aoki. Meet-in-the-middle preimage attacks on double-branch hash functions: Application to RIPEMD and others. In C. Boyd and J. M. G. Nieto, editors, *ACISP*, volume 5594 of *Lecture Notes in Computer Science*, pages 214–231. Springer, 2009.

[128] M. Schläffer and E. Oswald. Searching for differential paths in MD4. In Robshaw [121], pages 242–261.

[129] P. Schmidt-Nielsen. A distinguisher for reduced-round LUX. NIST mailing list (local link), 2009.

[130] B. Schneier, editor. *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*. Springer, 2001.

[131] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. *The Twofish encryption algorithm: a 128-bit block cipher*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[132] M. Stevens. Fast collision attack on MD5. *Cryptology ePrint Archive, Report 2006/104*, 2006. Available at `http://eprint.iacr.org/2006/104.pdf`.

[133] B. Su, W. Wu, S. Wu, and L. Dong. Near-collisions on the reduced-round compression functions of Skein and BLAKE. Cryptology ePrint Archive, Report 2010/355, 2010. Avalaible at `http://eprint.iacr.org/2010/355.pdf`.

[134] S. S. Thomsen. Pseudo-cryptanalysis of the original Blue Midnight Wish. In Hong and Iwata [65], pages 304–317.

[135] T. Tokita, T. Sorimachi, and M. Matsui. Linear cryptanalysis of LOKI and s^2DES. In J. Pieprzyk and R. Safavi-Naini, editors, *ASIACRYPT*, volume 917 of *Lecture Notes in Computer Science*, pages 293–303. Springer, 1994.

[136] U.S. Department of Commerce, National Institute of Standards and Technology (NIST). Secure Hash Standard. Federal Information Processing Starndard Publication 180-2, 2004.

[137] P. C. van Oorschot and M. J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *ACM Conference on Computer and Communications Security*, pages 210–218, 1994.

[138] K. Varıcı, O. Özen, and Çelebi Kocair. Sarmal: SHA-3 proposal. Submission to NIST, 2008.

[139] D. Wagner. The boomerang attack. In Knudsen [84], pages 156–170.

[140] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[141] X. Wang and H. Yu. How to break MD5 and other hash functions. In Cramer [40], pages 19–35.

[142] S. Wu, D. Feng, and W. Wu. Cryptanalysis of the hash function LUX-256. Available online, 2008.